Chapter 11

The BTeV Trigger

11.1 Introduction

The BTeV experiment includes a sophisticated trigger system that rejects 99.9% of light-quark background events, while retaining large numbers of B decays for physics analyses. The design of the trigger supports BTeV's goal of studying a broad range of B decays using many different B-tagging techniques. To be competitive with other projects engaged in B physics (including those that will run concurrently with BTeV and those that are planned for the future), BTeV aims to maximize the number of B decays available for physics analyses by taking advantage of the high-resolution three-dimensional tracking data provided by the pixel vertex detector, by using a consistent trigger strategy throughout all stages of the trigger system, by analyzing every bunch crossing to search for evidence of a B decay, and by deploying a trigger system that is fault tolerant and fault adaptive.

In this chapter, we begin with an overview of the trigger system in Section 11.2 and the BTeV trigger requirements in Section 11.3. We provide technical descriptions of trigger algorithms in Section 11.4. The sections that follow, 11.5 and 11.6, provide details of the trigger hardware, trigger R&D and simulation results. Discussions of supervision and monitoring for the trigger system, and the RTES (Real Time Embedded Systems) Project are presented in Sections 11.7 and 11.8, respectively. We conclude with a discussion of our production plan in Section 11.9.

11.2 Overview

The trigger system is crucial for the success of BTeV. An important feature of the BTeV trigger (one that distinguishes our trigger from ones used in other experiments), is that it finds B events with high efficiency in the first stage of triggering (referred to as Level 1) by taking advantage of the key property that differentiates B (and charm) particles from other types of particles, namely their characteristic lifetimes. To achieve this, BTeV must analyze every bunch crossing by performing track and vertex reconstruction to search for evidence of a particle-decay vertex within a few hundred microns to a few centimeters away from a primary interaction vertex. In practice, at Level 1 this is done by reconstructing all primary vertices and selecting events that have additional tracks with large impact parameters with respect to the nearest primary vertex. Other experiments [1] use a fairly simple ”first level” of triggering. These types of triggers are able to reduce the number of events so that subsequent trigger levels have more time to deal with the surviving events, but they also restrict the types of final states that are accepted by the experiment, thereby limiting physics analyses. Trigger strategies that require the presence of specific final-state particles, such as muons, or demand the presence of a few high-p_T hadrons, are examples of this. By avoiding these restrictive trigger strategies at Level 1 and by exploiting the characteristic lifetimes of B particles at the first and subsequent trigger levels, the BTeV trigger is able to maintain high efficiency for B events throughout the entire event selection process.

The trigger system consists of three levels: L1, L2, and L3. Each level contributes to the reconstruction of events, and successive levels impose more and more refined selection criteria to select B events and reject light-quark background events. The trigger is designed to run at an initial (peak) luminosity of $2\times10^{32}\mathrm{cm}^{-2}\mathrm{s}^{-1}$ with a 132 ns, 264 ns, or 396 ns bunch crossing interval, corresponding to an average of 2, 4, or 6 interactions per crossing respectively. The L1 trigger operates at the Tevatron bunch-crossing rate and is the most demanding part of the trigger system. It consists of the L1 pixel trigger, L1 muon trigger, and Global Level 1 (GL1). The L1 pixel trigger reconstructs tracks and vertices for every bunch crossing. It is able to carry out track and vertex reconstruction at the bunch-crossing rate because of the very high-quality, low-noise, three-dimensional tracking information provided by the pixel detector. The data from the BTeV spectrometer are read out and processed in a large number of parallel data pipelines so the total processing time for data associated with a particular bunch crossing can be far greater than the bunch crossing interval, but the L1 trigger must produce trigger decisions with a time-averaged rate that is less than the bunch crossing interval.

The L1 trigger reduces the data rate by a factor of approximately 50 by rejecting background events while retaining B events with high efficiency for the next trigger level, L2. The L2 trigger improves the track and vertex reconstruction by reviewing the pixel data used at L1, and by including additional pixel hits in reconstructed tracks if necessary. L2 can also access additional data, such as data from the forward-tracking system, to further improve and refine track and vertex reconstruction. L2 reduces the data rate by rejecting at least 90% of bunch crossings that are sent to L2. At L3 all of the data for a bunch crossing are available. We perform a complete analysis of the data. This is comparable to the offline analysis performed by other high-energy physics experiments. L3 imposes the selection criteria for the final trigger decision and rejects at least 50% of the bunch crossings sent to L3.

The BTeV three-level trigger system is shown in Fig. 11.1. The figure shows a box at the top that represents the one-arm BTeV spectrometer. Data are read out from front-end electronics and are sent to the first-level trigger (L1) and to Level-1 buffers. An L2/3 crossing switch, which is part of the data acquisition system (see chapter **??**), is used to route data

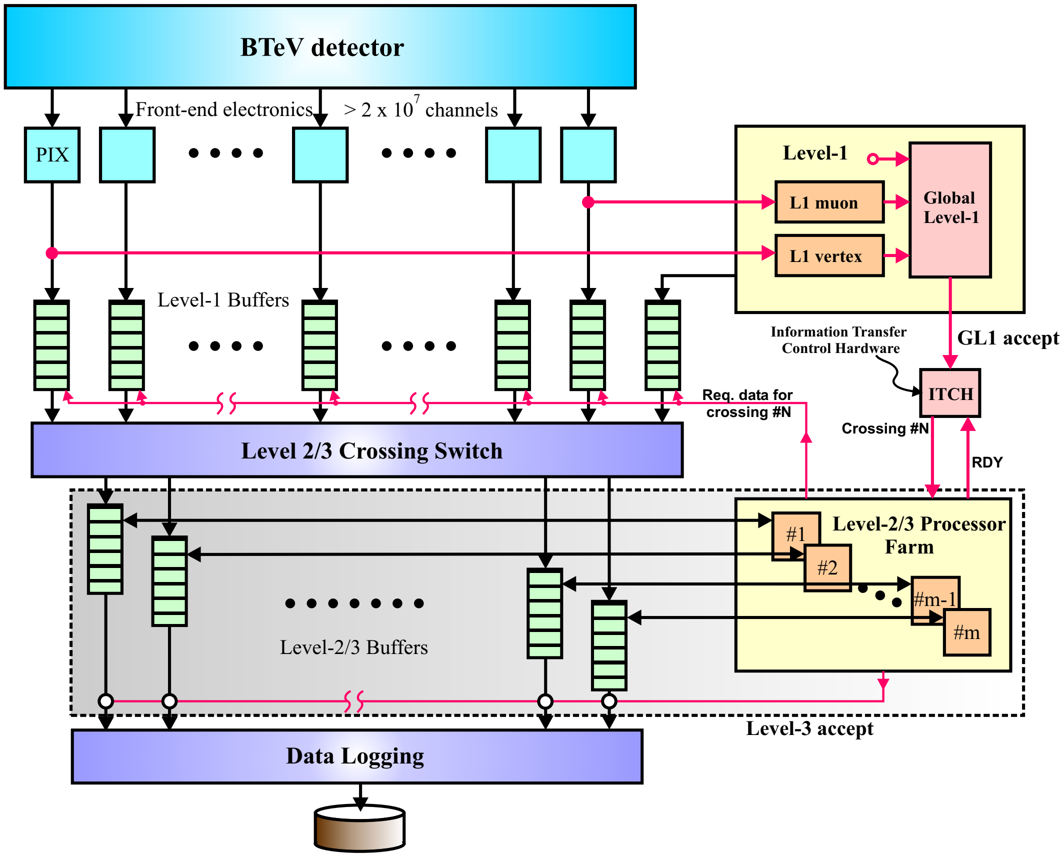

Figure 11.1: BTeV Three-Level Trigger Architecture

to the second and third level triggers for bunch crossings that satisfy the L1 trigger. Bunch crossings that satisfy L2 and L3 selection criteria are logged on archival media.

The trigger levels differ in the amount of time that is allotted for data processing, in the detector data that are available for processing (for L1, only the data from the pixel and muon detectors are available), and in the type of hardware used to process the data. L1 has the least amount of time available for processing and it uses a variety of hardware including field-programmable gate arrays (FPGAs) and general-purpose processors. L2 has more time to process data (due to data reduction that occurs at L1) and it uses a farm of Linux PCs. L3 has the longest amount of time available for processing and it uses the same farm of Linux PCs. Since the farm is used for both L2 and L3 it is referred to as the L2/3 trigger farm.

The distinction among the types of calculations that are performed at each trigger level are not rigid, and there is considerable flexibility in the design of our three-level scheme. This is especially true for calculations that are performed on data from the pixel detector. The pixel data, which are considered crucial for the success of the experiment, are used at all levels in our three-level scheme. The pixel data are analyzed by the L1 trigger in four stages: pattern recognition, track reconstruction, vertex reconstruction, and impact-parameter calculations that form the basis for the L1 trigger decision. In our baseline design the bulk of the pattern recognition for pixel data is performed by FPGAs, which excel at performing large numbers of rudimentary calculations in parallel. The remaining L1 calculations are performed by general-purpose processors (referred to as L1 farm processors or L1 worker nodes). The pixel data are also analyzed by subsequent trigger levels, L2 and L3, where general-purpose processors (Linux PCs) are used to perform trigger calculations. The flexibility of the overall design becomes apparent when one studies particular calculations and investigates how these calculations can be performed at different stages in the trigger architecture. For example, we have investigated two alternatives for pattern recognition that significantly reduce the processing time in the L1 worker nodes. The first is a "hash sorter" that can be implemented in an FPGA, thereby reducing the load on the L1 worker nodes. The second alternative also reduces the load on the worker nodes by moving calculations that complete the pattern recognition for pixel data from the worker nodes to FPGAs. Additional studies of this kind involving the optimization of the trigger architecture are underway and will help to improve the overall design of the trigger system.

The design of the trigger has evolved over time. Many of the changes that have been made were introduced to accommodate changes in the pixel detector, and in some instances modified trigger algorithms have led to significant improvements in the design of the pixel detector. An example illustrates the interdependence of the trigger and pixel detector and the importance of concurrent development of the two systems [2]. For the BTeV Proposal [3] the pixel detector was modified from three pixel planes per tracking station to two planes per station. The removal of an entire plane from each tracking station led to a reduction in cost, a reduction in material, and a pixel detector that would be easier to build. However, this change in the pixel detector was only adopted after it had been proven that a new L1 trigger algorithm was able to satisfy all trigger requirements without loss of physics capabilities. Other changes in the design of the pixel detector have been made, and each change was accompanied by considerable effort to modify trigger algorithms, repeat simulations, and tune algorithm parameters to maximize physics capabilities.

Finding a balance between available hardware and required physics computations has been an ongoing effort during the R&D phase of the BTeV trigger project. In the sections that follow we present our baseline design for the BTeV trigger system. This design has evolved over time, and will continue to evolve as new hardware becomes available. Some of the strategies that have been considered, evaluated, and abandoned (for the time being) are the following:

- We considered using general-purpose processors for the entire trigger system, but found that the cost would be prohibitive. In particular, we found that the pattern recognition for the pixel data required far too many compute cycles on a general-purpose processor. In our current design FPGAs are used for pattern recognition, and general-purpose processors are used for all other calculations.

- We investigated the use of different types of digital signal processors (DSPs) for the

L1 trigger. We studied the use of fixed-point DSPs for pattern recognition, and found that we required a large number of DSPs for pattern recognition. For several years our baseline design was based on floating-point DSPs that were used to perform track and vertex reconstruction and impact parameter calculations. The floating-point DSPs were recently replaced by general-purpose processors in our baseline design to take advantage of recent technological developments and the significant increase in processing power that we are able to obtain with general-purpose processors.

- We have studied other pattern-recognition algorithms that would be implemented largely with FPGAs. Our current algorithm (which is also based on an FPGA design) requires only a fraction of all pixel hits by finding two track segments for each particle trajectory: one track segment at the beginning of a trajectory (as the particle enters the pixel detector), and one track segment as the particle exits the pixel detector. Other algorithms that we have studied used all of the pixel hits associated with a particular track. One of the algorithms found three-station track segments throughout the pixel detector but suffered from significant increase in data volume as the data progressed through several stages in the trigger hardware.

We continue to explore alternatives to improve the design of the trigger system.

11.3 Requirements

This section describes the requirements for the BTeV trigger system. The requirements are based on detailed investigations of algorithms and technology believed to help the experiment achieve its physics goals while being both affordable and technically achievable. An understanding of the requirements presented in this section may require an understanding of details of the trigger system presented in subsequent sections in this chapter. We encourage the reader to refer to later sections for additional information.

11.3.1 Rate Requirements

The rate requirements for the trigger are determined by the running conditions of the Tevatron and the physics goals of the experiment.

- *L1 trigger rate:* The L1 trigger must make trigger decisions at the bunch-crossing rate every 132 ns, 264 ns, or 396 ns on average.

- *Number of interactions per bunch crossing:* The BTeV trigger must be able to handle an average of 2, 4, or 6 interactions per bunch crossing (depending on the bunch-crossing rate), and maintain good efficiency for B events accompanied by an average of 2, 4, or 6 minimum bias events, respectively.

11.3.2 Algorithm Requirements

BTeV requires the ability to study a broad range of B-decays including B_d, B_u, B_s, B_c, and all types of b-baryons. We intend to study these decays in a wide variety of final states including those that have only charged hadrons, those that have charged and neutral hadrons, photons and π^0s, electrons, and muons.

- *Trigger algorithm:* The BTeV trigger must base its decision on the characteristic properties of B decays so as not to limit the physics potential of the experiment. These properties must be determined well enough so that any needed corrections do not unduly limit the physics reach of the experiment.

11.3.3 Output Data Rate Requirements

Studies of the cost of data storage and retrieval have led to the conclusion that the output data rate should be no more than 200 MBytes per second. This corresponds to 2 PetaBytes per Snowmass year (10^7 seconds of running the experiment).

- *Output data rate:* The output data rate of the trigger must average to no more than 200 MBytes/sec at peak luminosity.

11.3.4 Rejection and Efficiency Requirements

The efficiency of the trigger depends on the particular parent B particle and its final state decay topology. The efficiency is defined for a particular analysis that is tuned to have acceptable signal to background ratio to achieve the physics goals of the analysis. The events that survive the analysis cuts represent the sample of events used to determine the trigger efficiency. The efficiency is defined as the fraction of these events that survive the trigger. Extensive simulations have shown that an efficiency greater than 50% can be achieved for most decay topologies with at least two charged hadrons associated with the B vertex, or the B vertex and an associated charm vertex. In cases with only one charged particle emerging from the B vertex and a non-charm decay such as a K_s, or a charm decay that has only one prong, the efficiency will be somewhat lower but should be at least greater than 20%. This determination of the trigger efficiency assumes 100% pixel efficiency and does not take into account the trigger livetime and uptime, which have their own requirements and are described in Section 11.3.6. Although we believe that pixel efficiencies close to 100% are achievable, we require that the trigger efficiency does not drop below 45% as long as the pixel detector is operating within its performance envelope.

Simulations have also shown that an average event size (after sparsification by the readout or front-end electronics) of less than 200 KBytes per bunch crossing is achievable at a 396 ns crossing time and an average of 6 interactions per bunch crossing. This represents the amount of data per bunch crossing for events that are analyzed by the L1 trigger. Further reduction in the event size can be obtained by eliminating some information associated with non-B

interactions, summarizing and condensing the remaining information, and possibly applying a compression algorithm somewhere between the trigger and data storage. The result is that we expect the average amount of data per bunch crossing written to archival storage to be approximately 80 KBytes at a 396 ns crossing time and an average of 6 interactions per bunch crossing. This implies an output rate of 2500 bunch crossings per second. In this document we give numbers for both a 396 ns crossing time (an average of 6 interactions per crossing) and for a 132 ns crossing time (an average of 2 interactions per crossing). At a 132 ns crossing time we would expect the average amount of data per bunch crossing written to archival storage to be about 50 KBytes, corresponding to an output rate of 4000 Hz. This implies that one must handle an output rate (see Section 11.3.3) of up to 4000 bunch crossings per second.

- *Output crossing rate:* The BTeV trigger must accept bunch crossings at a rate compatible with the maximum output data rate (see Section 11.3.3). It is expected that the trigger will accept no more than 4000 bunch crossings per second.

- *Efficiency:* The efficiency of the trigger must be greater than 50% (not including livetime or uptime, and assuming 100% pixel efficiency) for nearly all B decays having two or more charged particles emerging from the B or daughter charm vertex. The trigger efficiency must be greater than 45% when the pixel detector is operating within its performance envelope.

- *Single prong efficiency:* The efficiency for less well defined states such as those with a single prong and a K_s must be greater than 20%.

- *L1 rejection:* The L1 rejection must be greater than 98% of all bunch crossings.

- *L1 efficiency:* The L1 efficiency, as defined above, must be greater than 50% for B decays having two or more charged decay particles. The efficiency must be greater than 45% as long as the pixel detector is operating within its performance envelope.

- *L2 rejection:* The L2 rejection must be greater than 90% of bunch crossings sent to the L2 trigger.

- *L2 efficiency:* The L2 efficiency must be greater than 90%.

- *L3 rejection:* The L3 rejection must be greater than 50% of bunch crossings sent to the L3 trigger.

- *L3 efficiency:* The L3 efficiency must be greater than 95%.

11.3.5 Global Level 1 Requirements

Global Level 1 (GL1) refers to the hardware and software that combines results from all L1 triggers (such as the L1 pixel and L1 muon triggers) and data from front-end electronics to select the bunch crossings that pass the first level trigger. GL1 includes the Information Transfer Control Hardware (ITCH), which assigns L1 accepted bunch crossings to L2/3 worker nodes. The data that are sent to GL1 are referred to as trigger primitives. The trigger primitives are not required to arrive in any particular order. GL1 must decide when it has all the trigger primitives that it needs for a particular bunch crossing.

- *GL1 trigger rate:* GL1 must accept trigger primitives from L1 worker nodes and front-end electronics at the full bunch-crossing rate, and must produce trigger decisions every 132 ns, 264 ns, or 396 ns on average (depending on the bunch-crossing rate).
- *GL1 trigger list:* GL1 must be able to inspect the trigger primitives for a bunch crossing and test against a group of conditions, called a *trigger list*, to see if the crossing satisfies one or more of the conditions. It must apply a prescale factor to each crossing that has satisfied a particular trigger. GL1 must then OR the results to determine whether the crossing satisfies the L1 trigger for that list.
- *GL1 partitioning:* To support partitioning of the trigger and data acquisition system, GL1 must be able to maintain multiple trigger lists and must be able to arbitrate if a particular bunch crossing satisfies more than one trigger list.
- *GL1 data packet:* GL1 must create a data packet, or packets, for each accepted bunch crossing. The data packets contain information that specifies which trigger lists were satisfied by the crossing, and contain the data from all trigger primitives suitably merged. The data packets must be buffered within GL1 and/or in a Level-1 buffer so that the data are available to higher-level triggers and recorded as part of the data written to archival storage.
- *GL1 prescale:* For diagnostic and monitoring purposes, GL1 must select events according to a prescale scheme and declare them as accepted. GL1 must record that these events were selected in this manner in the GL1 data packets.
- *GL1 signals to DAQ:* GL1 must issue any signals that might be needed by the data acquisition system (DAQ) to delete bunch crossings from Level-1 buffers for crossings that do not satisfy the L1 trigger. Similarly, GL1 must issue any signals needed by the DAQ to preserve crossings that satisfy the L1 trigger so that they are available to higher-level triggers.
- *GL1 statistics:* GL1 must maintain statistics required for the diagnosis of problems, for calculating deadtime, and for monitoring luminosity. It may need to obtain information about the luminosity from another source to determine whether dynamic prescaling is appropriate, but the current goal is to have GL1 keep statistics that can be used to determine the luminosity.
- *GL1 throttling:* GL1 must contain a mechanism for dynamically throttling and/or prescaling some triggers so that the triggers with a higher priority are taken. It must

do this based on information concerning luminosity and the availability of processors and buffer memory.

- *GL1 latency:* In order to facilitate operation, it is permissible for GL1 to set a small "minimum L1 latency," which is guaranteed. This will set the time available for the formation of triggers and the generation of front-end trigger primitives. The absence of valid data from the front ends will not delay the trigger decision.

- *GL1 timeout:* In order to avoid problems with buffer memory and with GL1 itself, it is permissible to impose a "timeout" after which GL1 will make a decision based on whatever information it possesses. The system will accept some prescaled selection of crossings that fail to have all information available. These crossings are used for subsequent evaluation of the impact of these losses on the physics. If this situation is sufficiently rare, all such failures could be declared to pass the trigger. GL1 needs to record in the GL1 data packets that the trigger accepted a crossing due to a timeout, and must maintain statistics on the frequency and nature of such occurrences.

- *GL1 accounting:* GL1 must account for all bunch crossings ensuring that each was accepted, rejected, or reported as missing. This accounting must be done within an amount of time that allows corrective action to be taken if the rate of lost data becomes unacceptable.

- *GL1 assignment:* GL1 will assign each bunch crossing that satisfies a trigger list to one L2/3 processing node.

- *GL1 and run control:* GL1 must respond to Run Control commands and must be able to support partitioning (see Section 11.3.12).

11.3.6 Livetime and Uptime Requirements

The trigger performance can be affected adversely by factors both within the trigger and outside of it. Factors outside of the trigger system include an unusually "dirty" beam that produces extra background in the detector, badly imbalanced bunch populations, poorly performing or noisy detectors, etc. These "external factors" have the effect of reducing the "length" of the data pipeline as measured in bunch crossings and can therefore cause the front-end electronics to run out of buffer space during the trigger decision time, or can cause the trigger decision time to be longer than average due to noisier than average events. "Internal factors" include high failure rates for processors or bottlenecks in the data-transport network, which would prevent processors from being used efficiently. Any of these problems can cause the processing to fail to keep up with the bunch crossing rate and must inevitably result in data being discarded and therefore lost.

Terms used in this section are deadtime/livetime and uptime/downtime. During uptime the trigger is performing normal operations, processing incoming data with trigger algorithms and producing output. During downtime the system is unavailable. During livetime the trigger is performing normal operations on all assigned bunch crossings, processing normally within the time and bandwidth requirements. No data is left unprocessed, no data is lost, and no other system operation is hindered or impeded by the trigger operation. Deadtime occurs when data is permanently lost to the experiment due to overflows, timeouts, or errors.

- *Livetime:* With a suitably pipelined architecture, the trigger should impose almost no deadtime on the experiment. We set the livetime requirement based on factors *internal* to the trigger to be greater than 95%.
- *Uptime:* The trigger uptime must be greater than 95%.

11.3.7 Excess Capacity and Scalability Requirements

While extensive simulations have been and continue to be carried out to verify the trigger performance, it is impossible to predict exactly what reality will be like. It is therefore important that the trigger has extra capacity built in, and that the architecture be such that additional expansion is possible with incremental funding.

- *Capacity:* The trigger must have at least 50% extra capacity.
- *Scalability:* The initial implementation of the trigger architecture must permit an increase in capacity of at least a factor of 2 in processing and data throughput at every level, and a factor of 2 in the size of buffer memories.

11.3.8 Flexibility Requirements

The trigger architecture must be capable of including possible expansion to satisfy new physics goals that might arise in the future.

- *Flexibility:* It must be possible to add additional types of triggers.

11.3.9 Fault Tolerance and Security Requirements

Fault tolerance and redundancy must be designed into the architecture of the trigger system. Moreover, the trigger system must have adequate computer security for computing systems that are part of the trigger and are accessible from systems external to the trigger.

- *Component failure:* The trigger must continue operating, perhaps at reduced capacity and efficiency, in spite of the failure of a number of the processors, network connections, etc.
- *Trigger arbitration:* If the trigger system is having difficulty keeping up with the data rate, it must be able to drop less important triggers and calculations so that it can maintain high efficiency for the most important physics.

- *Purging:* The trigger system must be able to purge data when the data rate is too high. A mechanism that logs data loss due to purging must be implemented.
- *Timeouts:* Timeouts must be used to impose limits on excessive calculations that tie up resources. The system must monitor and record the frequency of timeouts as a warning and diagnostic tool.
- *Data duplication:* Within the trigger system, bunch-crossing data will be moved and processed without duplication.

11.3.10 Detector Performance Envelope

Detectors never perform perfectly. BTeV detectors have specific performance envelopes that must be achieved and maintained. The trigger must be able to achieve its efficiency and rejection goals when all of the detectors are within their performance envelopes. There should also be enough headroom for the trigger to degrade gradually if one or two detectors move slightly outside of their worst acceptable performance.

- *Performance envelope:* The trigger must achieve its efficiency and rejection requirements (see Section 11.3.4) when all detectors that affect the trigger performance are within their performance envelopes.

11.3.11 Supervision and Monitoring Requirements

The trigger supervision and monitoring system is interfaced to the global control and monitoring system for the BTeV experiment. It is responsible for the supervision and monitoring of all hardware in the trigger system. Examples of functions performed by the trigger supervision and monitoring system include programming for Field Programmable Gate Arrays (FPGAs), programming for L1 and L2/3 worker nodes, in-situ diagnostics and testing, performance monitoring, as well as status and error message reporting. In addition to performing these functions, the system must be able to receive commands from and report back to the BTeV control and monitoring system. It must record any unusual conditions or problems in the trigger and report them.

- *Supervision and monitoring:* The trigger supervision and monitoring system must receive commands from and report back to the BTeV control and monitoring system.
- *Read-back for programmable devices:* The supervision and monitoring system must be able to read data from programmable devices in L1, L2, and L3. This includes the programs, parameters, device configurations, status and error messages, any temperature and voltage measurements, as well as processed data at useful probe points in the data stream.
- *Initialization:* The supervision and monitoring system must be able to configure trigger hardware and software.

11.3.12 Support for Commissioning and Debugging

The trigger system must be able to support not only steady-state operation but also commissioning and runtime troubleshooting. It will be necessary to test new trigger algorithms and new trigger hardware (for example, burning in a new set of processors). During commissioning, several detectors may be running almost autonomously for debugging or calibration purposes. The trigger system must be able to support partitioning. This means that it must simultaneously provide several different and independent triggers for different subsystems and allocate data to different groups of processors.

- *Partitioning:* The trigger system must be able to support partitioning.
- *Testing:* All programmable devices that are part of the trigger must be testable in-situ.
- *Database:* A resource identification and configuration database must be used to keep track of hardware and software used in the trigger system.

11.3.13 Software Requirements

The "software" for the trigger system refers to algorithms, specialized operating system code, as well as diagnostic and testing code developed for programmable devices in the three trigger levels, Global Level 1, and the supervision and monitoring system. The software includes FPGA firmware, software developed for specialized processors, and software developed for general purpose processors. In addition, the Real Time Embedded Systems (RTES) project will contribute software that will provide error handling, reporting and recovery techniques to enhance the reliability and robustness of the system. This software will be integrated at several levels into the trigger system, but it will only be allowed to use a small amount of the available processing resources. All of the software must be developed using standard software tools that allow version tracking, assist code reviews, and help with maintainability.

- *Error detection:* Processes that regularly verify code purity and run standard datasets for testing purposes must be part of the development process, and must be used for testing the trigger after the development of the trigger has been completed.
- *Software Database:* Trigger parameters and constants (such as prescale factors, geometry and alignment constants) must reside in a database so that the particular values used to process data can always be identified.
- *Software repository:* All software must reside in a software repository that must be used to keep track of different versions of the software during development.
- *Version control:* The version numbers of software used to process data must be managed in such a way that the particular version that was used to process the data can always be identified and reproduced.

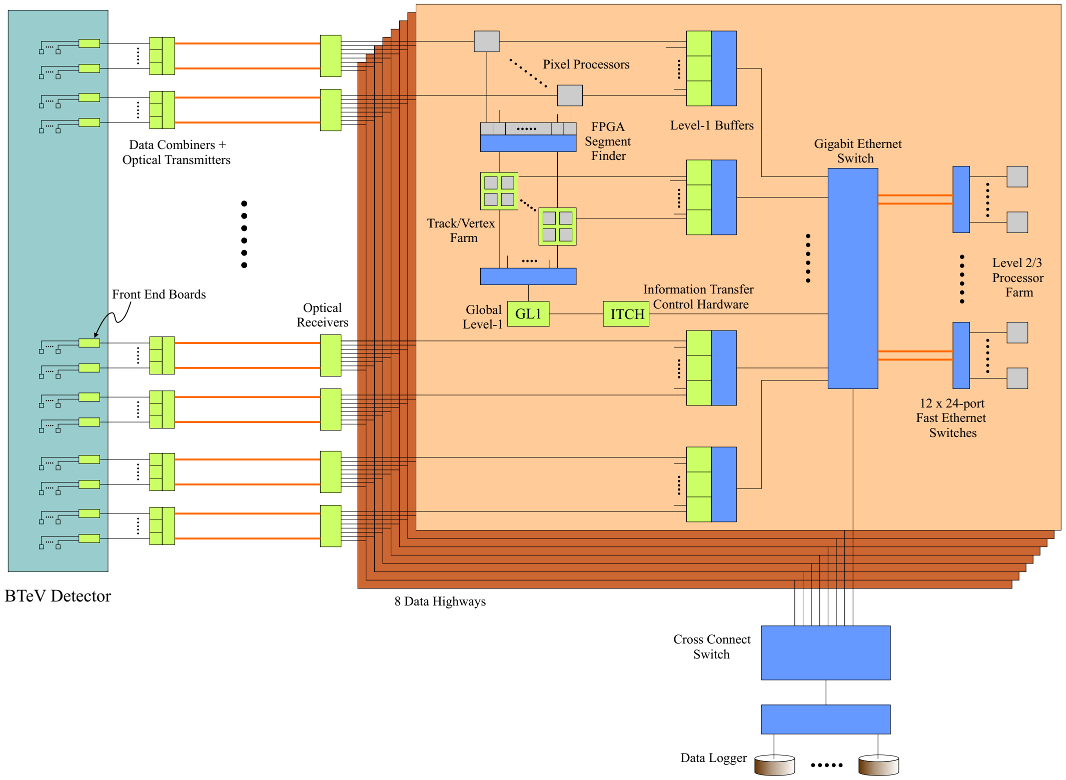

Figure 11.2: BTeV Three-Level Eightfold-way Trigger Architecture.

- *Processing resources:* RTES error detection, recovery and reporting software, as well as other types of control and monitoring software will use less than 10% of the processing resources of any trigger system unit on which it is installed.

11.4 Technical Description and Implementation

11.4.1 Overview

We begin this section with an architectural overview of the BTeV trigger system, describing the basic components of the 3-level trigger system and how they interact with each other [15]. We will also show how the trigger system integrates into the larger picture of the BTeV data acquisition system. For more details on various components of the data acquisition system relevant to the trigger system, please refer to Chapter **??**. The overview will then be followed by more detailed subsections focusing on each of the major trigger components: L1 pixel, L1 muon, GL1, and L2/3. These subsections will describe the algorithm and hardware aspects of each component. The final subsection will describe completed and ongoing R&D work on each of these trigger components.

The trigger system consists of five subsystems:

- *L1 pixel trigger:* The primary trigger for the experiment. It receives data from the pixel detector.
- *L1 muon trigger:* The trigger that selects J/ψ and prompt muon events and is used to calibrate the L1 pixel trigger. It receives data from the muon detector.
- *GL1 trigger:* Global Level 1 (GL1) receives data from the L1 pixel trigger, L1 muon trigger, and front-end electronics. It selects the bunch crossings that pass the first level trigger. It also includes the Information Transfer Control Hardware (ITCH) that assigns accepted bunch crossings to L2/3 worker nodes.
- *L2/3 trigger:* The L2/3 trigger consists of the L2 and L3 algorithms running together on a processor farm (L2/3 trigger farm). A bunch crossing that is analyzed by a particular processor and satisfies L2 selection criteria is processed by L3 trigger algorithms without requiring a transfer to a different processor. Bunch crossings that fail are eliminated. The distinction between L2 and L3 is that L3 has access to all of the data for a particular bunch crossing, while L2 operates on a subset of the data.
- *Supervision and monitoring:* The supervision and monitoring system (not shown in Fig. 11.1) consists of separate subsystems for each of the four trigger subsystems: PTSM, MTSM, GL1SM, and L23SM for the L1 pixel, L1 muon, GL1, and L2/3 triggers, respectively.

BTeV's 3-level trigger architecture is shown in Fig. 11.2. Data from the detector's front-end electronics are sent at the full crossing rate from the collision hall to the counting room via high-speed optical links. Optical receivers in the counting room distribute the data for each bunch crossing uniformly to one of eight parallel data paths called "highways", each of which forms a complete and independent 3-level trigger system. The full data rate from the detector is $\sim$500 GB/s assuming a crossing rate of 2.5 MHz with an average of 6 interactions/crossing, and an average event size of 200 KB. The eightfold highway architecture reduces this full data rate to 62.5 GB/s into each highway allowing the use of low-cost components such as commercially available Ethernet switches. The choice of *eight* parallel data paths (as opposed to some other number) is used for the baseline design. The design is flexible and modular enough to reconfigure to other choices of numbers of parallel data paths if needed, due for example to possible changes in running conditions or to any changes based on bandwidth studies or cost considerations.

Note that in this document the default trigger and data rates we give are for a 396 ns crossing time (2.5 MHz crossing rate) and an average of 6 interactions per bunch crossing. The BTeV Trigger system has been designed to also handle a 132 ns crossing time so some rates are also given for a 132 ns crossing time (7.6 MHz crossing rate) and an average of 2 interactions per bunch crossing.

In each highway, data from all detector subsystems are temporarily stored in Level-1 buffers as trigger decisions are made. The Level-1 buffers consist of commodity SDRAM with enough memory to buffer $\sim 2{\times}10^6$ crossings in total, or about 250,000 crossings in each highway corresponding to $\sim$800 ms (over 3 orders of magnitude more than the average L1 processing time) of L1 trigger decision time. Aside from going to the Level-1 buffers, data from the pixel and muon detector are also sent to the L1 pixel and muon triggers, respectively. Since the hardware for the L1 muon trigger is very similar to the L1 pixel trigger hardware, only the latter is shown in Fig. 11.2 for simplicity.

Before they are sent to Level-1 buffers, pixel detector data pass through pixel preprocessors that perform various operations on the data such as time-stamp expansion, pixel hit clustering, and x-y coordinate lookup. The processed data are then sent to the Level-1 buffers and an FPGA based segment tracker that executes the segment finding stage of the L1 trigger algorithm (see Section 11.4.2). Inner and outer track segments from the same beam crossing are routed by a network switch to a single worker node in the L1 Farm, where the track and vertex finding stage of the L1 trigger algorithm is performed. Results from each node are saved in Level-1 buffers that reside on the worker nodes, while summarized trigger results are sent to a GL1 processor responsible for the ultimate L1 trigger decision. These decisions will be stored as a list of accepted crossing numbers in the Information Transfer Control Hardware (ITCH) which broadcasts accept messages to all Level-1 buffers.

The L1 muon trigger is based on the same hardware used for the track and vertex farm of the L1 pixel trigger. The GL1 processors combine L1 muon trigger decisions with those from the pixel trigger to form the final L1 trigger decision.

The Level-1 buffers are grouped into 32 subsystems, each of which feed a port in the highway switch consisting of a commercial 64-port Gigabit Ethernet switch. 12 pairs of ports from this switch feed the paired Gigabit uplink ports of a dozen 24-port fast Ethernet fanout switches. The fanouts, in turn, feed data to 96 commodity dual-CPU Linux-PC's that make up the L2/3 processor farm in each highway where the DRAM in each PC functions as an L2/3 buffer.

L2/3 nodes with available processing resources send requests for data to the ITCH which responds by assigning one or more accepted crossing numbers to that node. Once it receives its assignment, the L2/3 node sends a request to a subset of the Level-1 buffers which respond by sending their data to that node. The actual number of crossings assigned for each transfer can depend on the running conditions and, therefore, an optimal transfer packet size will be selected for each transfer. All requests and data transfers between the L2/3 farm and the Level-1 buffers and the ITCH are routed through the highway and fanout switches. Upon receiving the data, the L2/3 node executes the L2 trigger algorithm. If the event passes L2, the L3 algorithm is executed in the same node. L3 will be similar to the offline reconstruction, doing refined tracking and particle identification to improve upon the results from L2. L3 will perform some data reduction by dropping some raw data, summarizing some physics data, and, if necessary, performing some amount of data compression.

If the event passes the L3 trigger, the processed results are propagated back up the fanout

Trigger Level	Trigger Parameters				
	Input	Output	Reduction	Processing	Minimum #
	rate	rate	Factor	Time (ms)	CPU's
Level 1	2.5 MHz	50 KHz	50	0.08	202 (@100%)
	500 GB/s	12.5 GB/s			336 (@60%)
	200 KB/evt	250 KB/evt			
Level 2	50 KHz	5.0 KHz	10	5	250 (@100%)
	12.5 GB/s	1.25 GB/s			416 (@60%)
	250 KB/evt	250 KB/evt			
Level 3	5.0 KHz	2.5 KHz	2	134	672 (@100%)
	1.25 GB/s	200MB/s			1120 (@60%)
	250 KB/evt	80 KB/evt			

Table 11.1: Summary of trigger and data rates at each of the three levels of the BTeV trigger system. The numbers are for 396 ns bunch crossings with an average of 6 interactions per crossing. The processing time is per CPU per event (crossing). The actual processing is essentially deadtimeless with the use of pipelines.

and highway switches to an external cross-connect switch that routes accepted events from all 8 highways to a small cluster of data-logging nodes for archival storage.

The L1 trigger rejects at least 98% of all crossings, reducing the input data rate of 62.5 GB/s into each highway to 1.6 GB/s into the L2/3 farm. The L2+L3 trigger rejects 95% of the L1-accepted crossings so that the data rate out of L2/3 is a mere 25 MB/s, with a reduction of a factor of $\sim$3 in the data size per bunch crossing. This 25 MB/s out of each highway amounts to a total of 200 MB/s going into the data-logging nodes.

Table 11.1 summarizes the trigger and data rates at each of the three levels of the BTeV trigger system for 396 ns bunch crossings with an average of 6 interactions per crossing. The processing time (latency) per CPU is indicated in the table. The latencies imply a certain minimum number of CPUs for each trigger level. The table gives these minimum numbers of processing units when one assumes 100% usage of a processing unit for the trigger application. The baseline design is based on a larger number of processing units with a more realistic assumption for CPU utilization.

11.4.2 L1 Pixel Trigger

11.4.2.1 Algorithm

The L1 pixel trigger employs a two-stage algorithm that consists of a segment-finding stage (pattern recognition) followed by a track and vertex finding stage. For a detailed description of the algorithm, please refer to Ref. [2, 5]. The pixel vertex detector consists of an array of 30 stations of silicon pixel planes perpendicular to the beamline and distributed over about

125 cm along the interaction region. Each station contains two planes: one with the pixels oriented so the narrow pixel dimension is horizontal (bend plane) and one with the narrow pixel dimension in the vertical direction.

First Stage: Segment Finding

The segment finding stage, which is sometimes referred to as the pattern recognition stage, finds the beginning and ending segments of tracks in two separate regions of the pixel planes, an inner region close to the beam axis and an outer region close to the edge of the pixel planes. It restricts the search for the beginning and ending segments to these inner and outer regions, respectively. Since segments are found using hit clusters from three adjacent pixel stations in the defined regions, beginning segments are referred to as inner triplets while ending segments as outer triplets.

The search for inner triplets begins by looking for the two most upstream hits which will be referred to as inner doublets. These doublets are found using all combinations of hit clusters within defined regions of the bend plane of two adjacent stations. The upstream hit which is restricted to the inner region is paired with a hit from the downstream station and the straight line segment joining the two hits is projected upstream by a distance equal to the separation distance between pixel stations. If the x-y coordinates of the projection upstream come within or close to the edge of the beam hole in the center of the pixel station, the segment is then projected downstream to predict the position of a hit in the bend plane of the third adjacent station. If a hit cluster lies within a given distance of the predicted position, the algorithm then checks to see if there are at least two confirming hits in the non-bend planes of the three adjacent stations. If all these requirements are satisfied, the hits are then grouped together as an inner triplet.

The search for exterior triplets proceeds in a similar fashion. Exterior doublets are found using combinations of hits from the bend plane of two adjacent stations where the downstream hit is restricted to the outer region of the pixel plane. The straight line segment joining the two hits is projected upstream by one station and the x-y coordinates required to lie within pixel plane boundaries. The segment is then projected downstream by one station and the x-y coordinates required to lie close to or beyond the edge of the pixel plane. For doublets that meet these requirements, the upstream projection is used to predict the position of a hit in the third station upstream of the doublet. If a hit cluster lies within a given distance of the prediction, the three hits are grouped together as an outer triplet without requiring confirming hits in the non-bend planes of the three stations.

Second Stage: Track and Vertex Finding

Once the inner and outer segments are found, the algorithm proceeds to the second stage which is referred to as the track and vertex finding stage. This stage consists of two separate phases. The first phase, referred to as the segment matching phase, attempts to match inner to outer triplets belonging to the same track. To do this, inner triplets are projected downstream based on their slopes in the non-bend plane and their curvatures (slope change) in the bend plane. Outer triplets are matched to inner triplets to form complete tracks based on their proximity to these projected trajectories.

After the segment matching phase, the algorithm then proceeds to the vertex finding phase. It begins this phase by processing the tracks found from the previous phase, calculating their momentum from the curvature and knowledge of the B-field and calculating its transverse distance from the beam axis. It then loops over all tracks with $p_T < 1.2$ GeV/c and which appear to originate close to the beamline to search for primary interaction vertices. The first track in the loop is considered a seed track for a new cluster of tracks. The second track is then paired with the first to see if they form a cluster by calculating their point of closest approach (vertex position) and requiring the track to come within a given distance of this point. If the track meets this requirement, it is added to the cluster. If not, the track is used as a seed for a new cluster. This procedure is repeated for all subsequent tracks. If more than one cluster exists, a new track is attached to that cluster for which its proximity to the vertex position is smallest. At the end of the search, the cluster with the most number of tracks is then considered a primary interaction vertex. The algorithm loops through all tracks attached to the non-primary clusters to see if they fit the primary interaction vertex attaching them to this vertex if they do.

All tracks attached to the primary vertex are then tagged and the whole procedure described above to search for primary vertices is repeated using the remaining tracks from the non-primary clusters. Once all the primary vertices are found, the algorithm searches for detached tracks by looping over all tracks not attached to any primary vertex and calculating its impact parameter from each primary vertex. The detached track is then assigned to that primary vertex for which its impact parameter is smallest. An L1 vertex trigger accept is generated if there are at least 2 detached tracks in the same arm of the BTeV detector, associated with the same primary vertex, and satisfying the following criteria: $p_T^2 \geq 0.25$ (GeV/c)^2, $b/\sigma \geq n$, and $b \leq 2$ mm where b is the impact parameter, σ is the error in b, and n is a preset value ranging from 2-7.

11.4.2.2 Hardware

The hardware architecture of the L1 pixel trigger hardware is shown schematically in Fig. 11.3. As described above, data from the pixel detector front end is sent to FPGA based pixel processors that group individual pixel hits into clusters and translate the row column information into x-y coordinates. Hit clusters from three neighboring pixel stations are then routed to FPGA based hardware that finds the beginning and ending segments of tracks in the segment finding stage of the L1 trigger algorithm.

The segments or triplets found in this stage are sorted by a network switch according to crossing number and routed to a worker node in the L1 Farm. The worker node performs the track and vertex finding stage of the L1 algorithm. With an average processing time of 200 μs for the segment matching and vertex finding stage and an interaction rate of 2.5 MHz, a total of ~500 CPUs (or 250 dual-CPU worker nodes) will be required for the track and vertex farm in order to examine every single bunch crossing with a 50% duty time on

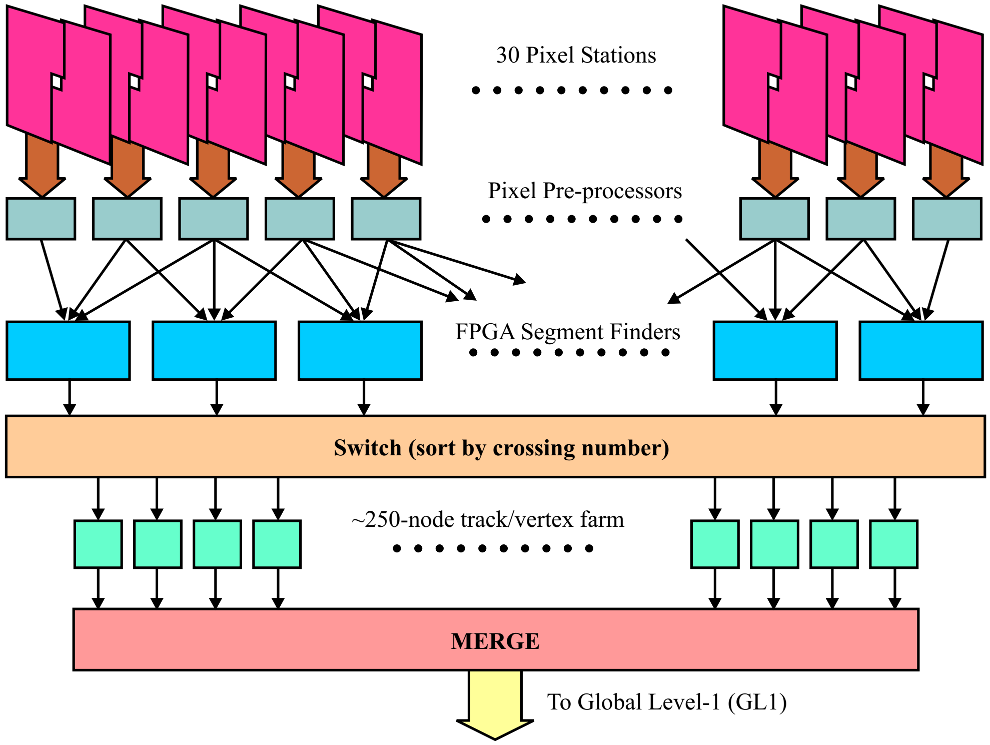

Figure 11.3: BTeV L1 Pixel Trigger Hardware Architecture

each CPU. Processed results are sent to Level-1 buffers while trigger summaries are sent to GL1 processors.

More details on the L1 pixel trigger hardware are provided in subsequent sections where we describe the completed and ongoing R&D work on the L1 trigger.

11.4.3 L1 Muon Trigger

11.4.3.1 Introduction

The purpose of the L1 muon trigger is to look for tracks in the muon detector consistent with muons originating in the interaction region. The main jobs of this system will be to identify dimuon events, in our quest to collect a large sample of B decays containing J/ψ's, and to help calibrate the L1 pixel trigger.

11.4.3.2 Geometry

The layout of the muon system is shown schematically in Fig. 11.4. The entire system is segmented into equivalent octants. Each of the three muon detector stations contains four detector planes, referred to as views in what follows. Each view (per octant) is composed of 12 planks, each containing 32 tubes. Of the four views within each station/octant, two

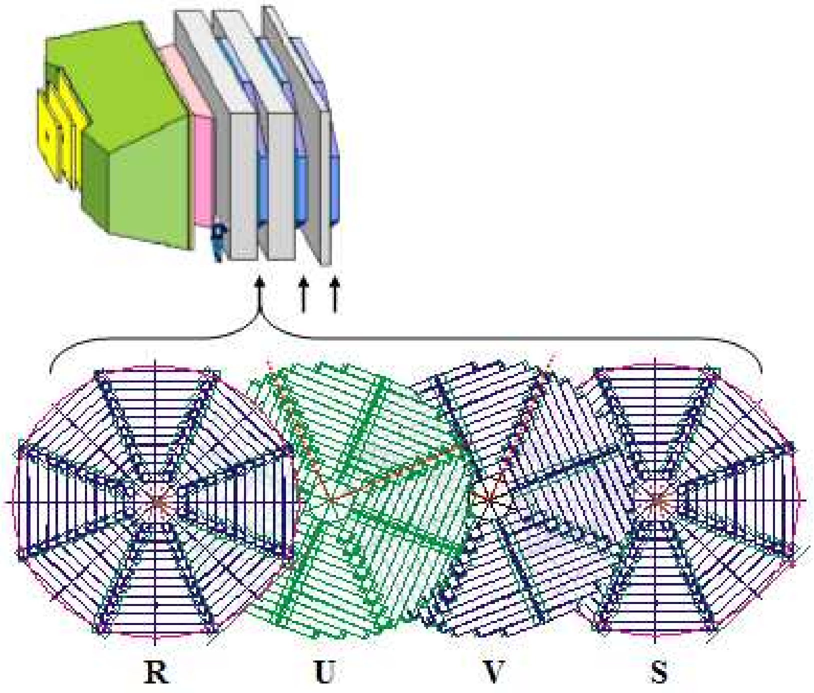

Figure 11.4: Schematic view of the BTeV muon system. Divided into octants, each of the three stations contains four views of detectors (R,U,V,S). Each view (per octant) contains 384 tubes in 12 planks.

views (R and S) are oriented perpendicular to the radial direction, and the two other views (U and V) are rotated ±22.5^o with respect to these.

11.4.3.3 Algorithm Overview

The purpose of the muon trigger is to identify muon tracks originating in the BTeV interaction region, and is accomplished by examining patterns of hit proportional tubes in the muon detector. The muon trigger system processes information from each octant independently, and within an octant, tracking is done independently on each of the four views.

To minimize combinations (i.e. processing time), adjacent hits within each view are sparsified by a simple algorithm that keeps only the "most central" tube (i.e. if tubes 11,12, and 13 are all asserted, the sparsification will keep only tube 12. If tubes 11 and 12 are asserted, only tube 11 is kept).

In order for track detection to be possible in a given octant/view, a track must leave hits in all three stations of that octant/view. The example illustrated in Fig. 11.5 shows a muon track leaving hits in all three R views of the uppermost octant.

Each muon which hits a specified octant/view in all three stations can be assigned a "co-ordinate" in the 3-dimensional space of this view. For example, the co-ordinate of the

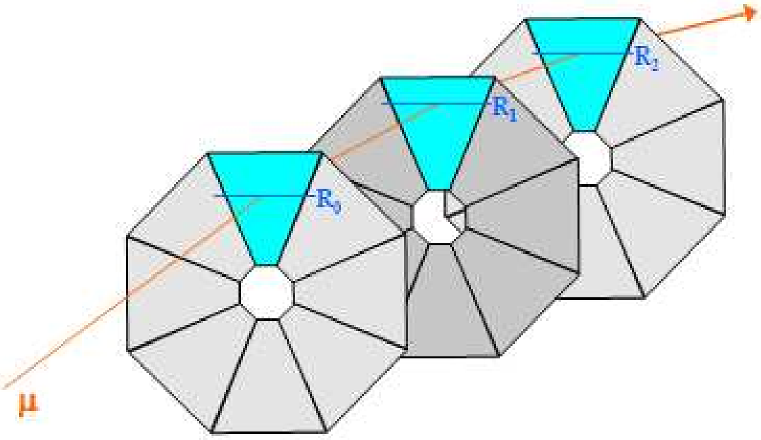

Figure 11.5: A muon track leaves hits in the R view of the top octant of all three muon detector stations.

track in Fig. 11.5 is (R0,R1,R2), where R0 is the number of the proportional tube hit in station 0, etc. In what follows, the “R” view will be used as an example, but the identical approach is used to define tracks in the “U”, “V” and “S” views as well.

Monte Carlo generated muons can be used to study the distribution of (R0,R1,R2) in the 3-dimensional R-space. The blue dots in Fig. 11.6[a] are the R-space coordinates (all octants combined) of about 2200 good muon tracks. These points lie in a very well defined plane — a fact that is more evident when examining Fig. 11.6[b], which is simply a rotated version of Fig. 11.6[a]. Note that the tubes are numbered from outside in: tube 0 is the outermost and tube 384 is the innermost in each view.

The equation of this plane can be found using a simple linear fit to the blue (good muon) points, and the perpendicular distance d to this plane from a given point (R0,R1,R2) can be easily calculated. Fig. 11.7 shows the d distribution for good muon events.

Given the minimal width of this distribution (the standard deviation of the entries in Fig. 11.7 is 1.5 tubes), we conclude that a simple plane in “R-space” does indeed provide a very good description of the (R0,R1,R2) coordinates of good muon tracks.

Although the results shown are for the R-view, nearly identical results are found when the same procedure is followed for the U, V and S views, hence we assume that tracking will be done in all 4 views using the same algorithm (albeit slightly different plane equations etc). For each view we define the normalized distance $D = d/\sigma$, where d is as described above and σ is the standard deviation of the d distribution for each view (i.e. the width of the peak in Fig. 11.7 for the case of the R view).

Cutting on D provides good rejection of minimum bias events because (R0,R1,R2) combinations from these do not tend to cluster around the “good muon plane” discussed above. This can be seen by examining the red points in Fig. 11.6[a] and Fig. 11.6[b], which indicate,

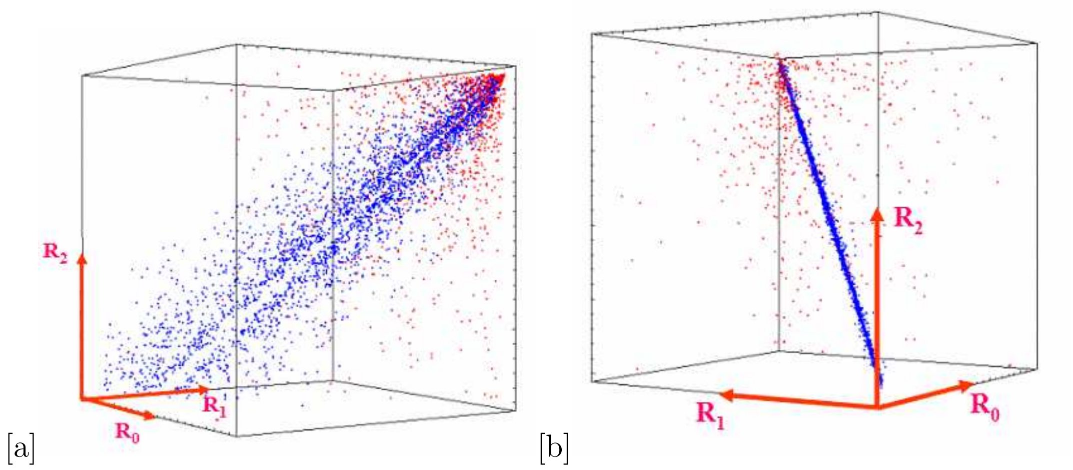

Figure 11.6: [a] Plot of (R0,R1,R2) for each of 2200 good muon tracks (blue dots) and the closest (R0,R1,R2) co-ordinate for each of 4300 minimum bias events (red dots). All octants are combined. [b] Plot rotated to illustrate that all of the good muon tracks (blue dots) result in (R0,R1,R2) coordinates that lie on a well defined plane.

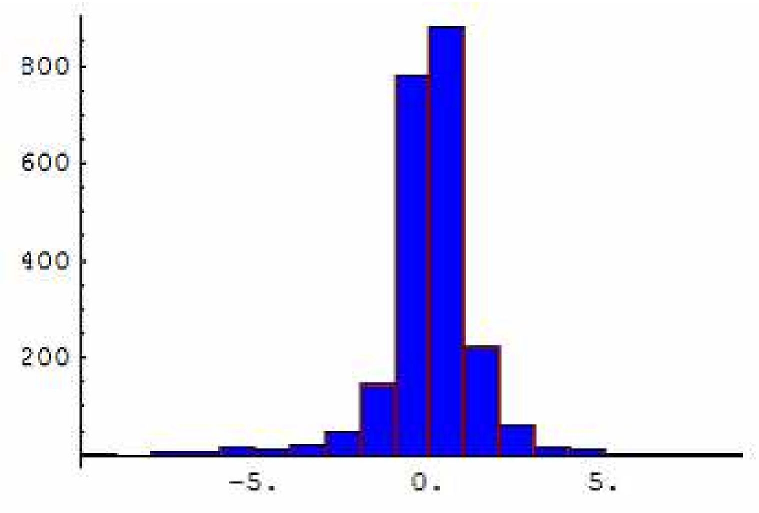

Figure 11.7: Distribution of distances *d* of each of the 2200 good muon (R0,R1,R2) coordinates from the plane defined by the best fit to the set of all such points, measured in tube number units.

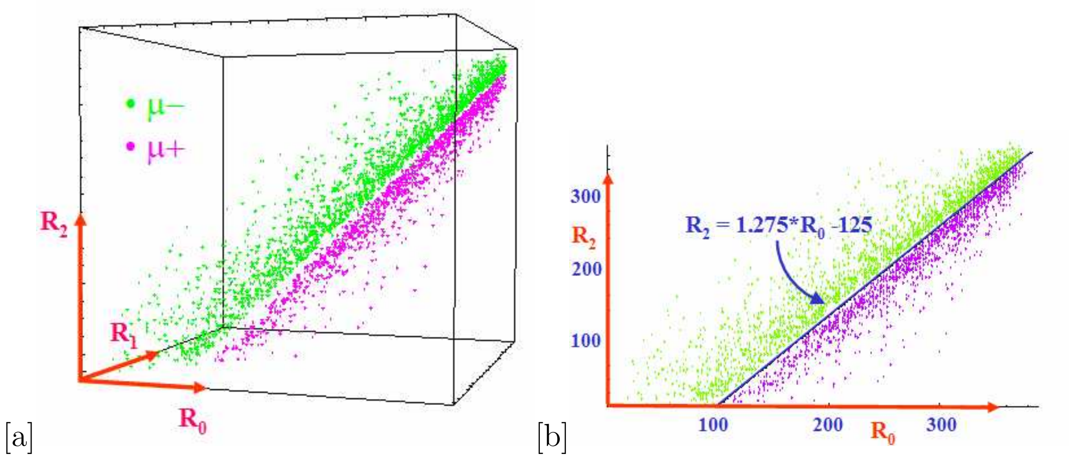

Figure 11.8: [a]Plot of (R0,R1,R2) for each of 1100 positive muon tracks (purple points) and 1100 negative muon tracks (green points). All octants are combined. [b]Plot of R2 vs R0 for each of 1100 positive muon tracks (purple points) and 1100 negative muon tracks (green points). All octants are combined. The blue line is a fit to the "gap" between positive and negative tracks.

for a sample of 4300 minimum bias events, the (R0,R1,R2) combination for each that is closest to the good muon plane.

Our strategy is thus to cut on D for each muon candidate, the tightness of this cut being a trade-off between efficiency and rejection (more on this below). Although not shown, the same discrimination is evident when examining the U-space, V-space, and S-space distributions.

11.4.3.4 Measuring Charge and Momentum

Upon closer examination of the good muon events (blue points) in Fig. 11.6[a], we find that useful kinematic information can also be extracted by simply considering the location of each point (i.e. track) within the plane discussed above. This is illustrated in Fig. 11.8[a], where the good muon tracks of Fig. 11.6[a] have been re-plotted such that the charge of each track is indicated by its color.

The charge separation is very clear, and indeed, nearly perfect charge identification can be obtained by considering only the (R0,R2) projection of this distribution. In other words, the correlation between the radial location of a track in the first and last stations gives good information about whether the track was bending inward or outward, which makes perfect sense. Fig. 11.8[b] shows the R2 vs R0 projection for the same events plotted in Fig. 11.8[a], along with the best fit line we used to determine the charge of muon candidates when simulating the dimuon trigger algorithm discussed below. Again, more or less identical results are found when examining the U, V and S views.

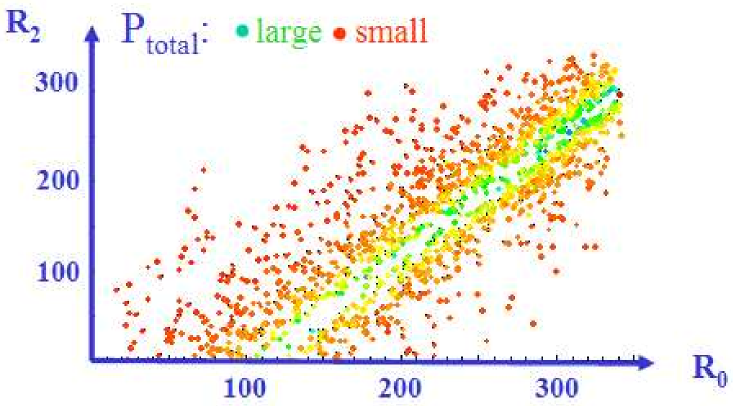

Figure 11.9: Plot of R2 vs R0 for 2200 good muon tracks. The color of the points indicate track momentum (red - small, green - large). All octants are combined.

Determining the charge of a track candidate is equivalent to measuring the sign of its curvature. The gap between positive and negative tracks that is evident in Fig. 11.8[a] and Fig. 11.8[b] is thus where tracks with zero curvature (i.e. infinite momentum) would live. We might therefore expect that the perpendicular distance of a point from this line is a measure of its curvature, hence inversely proportional to its momentum. Fig. 11.9, where the color of each dot indicates the total momentum of the good muon candidate it represents, gives qualitative evidence of this. No attempt has yet been made to use momentum discrimination in the muon trigger algorithm, although R&D to this end is planned.

11.4.3.5 Dimuon Algorithm

So far we have discussed the methodology for finding and determining the charge of single muon tracks. The thrust of the muon trigger will be to identify dimuon events, and the algorithm we have developed to accomplish this is discussed next.

Our current approach for triggering on $J/\psi \rightarrow \mu^+\mu^-$ is to look for events that contain at least one positive and one negative muon candidate, as illustrated in Fig. 11.10.

In each octant, the (R0,R1,R2) combination having the smallest absolute value of D is compared to a pre-determined D "cut", and, if it passes, the track-candidate's charge is determined by examining (R0,R2) as discussed above. The same algorithm is used for all four views in an octant (R, U, V and S), and for each view the output from an octant is NO (no track found), POS (positive track found) or NEG (negative track found). If the charge determination within an octant is ambiguous (i.e. if different views give different answers), then the octant is allowed to count for either (but not both) charges required by the dimuon trigger.

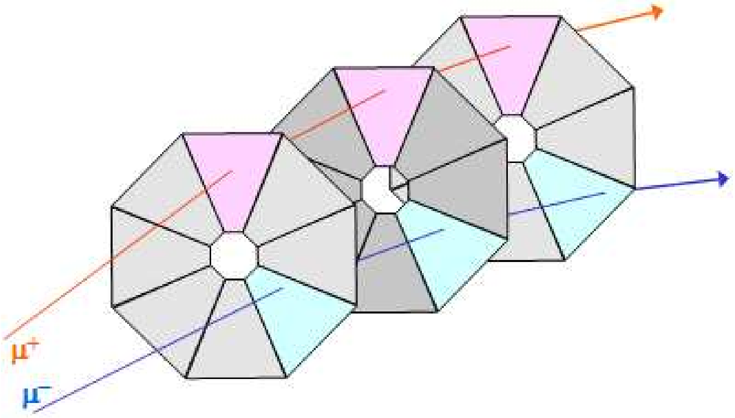

Figure 11.10: Schematic view of an event that would pass the dimuon trigger algorithm.

11.4.4 Global Level 1 Trigger

The picture of the L1 trigger that we have presented is a simplified one involving only the vertex and muon triggers. The actual L1 trigger is more complex and involves a variety of different triggers. This complexity is managed by the Global Level 1 Trigger (GL1).

First, both the vertex and muon triggers themselves can consist of several different selection criteria. For the vertex trigger, for instance, we have described a final selection that requires a minimum number of tracks to miss the primary vertex by a given number of standard deviations. In fact, there will be a variety of such triggers, some accepting events with a few tracks with large detachments and others accepting events with more tracks that have smaller detachments. We will also record a sample of events that would have failed the trigger requirements in order to understand how the efficiency of the trigger "turns on." This last group of triggers will be prescaled.

Second, there will be triggers that combine information from more than one detector at GL1. For example, a single-high-p_t-muon trigger would be interesting but may have too high a rate. However, the GL1 trigger could accept a high-p_t single-muon trigger if the event also satisfies a relaxed vertex requirement — perhaps one track with a large impact parameter. (At L1, one would not know that the high-p_T muon corresponded to the single high-impact-parameter track).

Third, there will be special triggers. One example will be a variety of minimum- and low-bias triggers that will be heavily prescaled. Another example will be special alignment and calibration triggers. We will collect special triggers or use minimum-bias triggers to do the quasi-real-time alignment of the pixel detector.

The main physics trigger will not be prescaled. GL1 will have the ability to prescale other less important physics triggers and calibration triggers. GL1 will also have the ability to adjust the prescale factors dynamically. For example, we will reduce prescale factors on some of the triggers as the luminosity falls. We will also increase the number of alignment triggers taken at the beginning of a store and then reduce them once enough events have been collected to establish initial alignment constants for the store.

The operation of the GL1 trigger is as follows. GL1 receives "trigger packets" from each trigger processor. These packets contain "trigger primitives" which are the data items from which GL1 makes the trigger decision. The packets, which have a format known to GL1, arrive asynchronously. GL1 buffers these packets until it receives all the packets it is supposed to receive for a crossing, or until a timeout occurs for that crossing. In normal operation, as soon as all packets are received, GL1 extracts the primitives and generates all the various triggers from truth tables that have been downloaded. It then applies the appropriate prescale to each trigger and takes the OR of the result. If any of the triggers is satisfied, it issues an L1 accept. This results in the event data being transferred to the output buffers of the Level-1 buffers for eventual transfer to an L2/3 node. If no trigger is satisfied, GL1 issues an L1 reject. This results in the Level-1 buffers being freed to be used for other events. In general, the trigger processors, which are exchanging messages with GL1, will have timeouts that are less than the GL1 timeout. If the allowed time for the arrival of an L1 trigger primitive expires before a trigger packet from a processor arrives, an error flag will be generated. A prescaled sample of these events can then be recorded for further analysis and diagnostics. The rest of the GL1 decision proceeds as normal.

To minimize the number of designs that must be developed and maintained, GL1 will be implemented using hardware identical to the hardware used for the L1 Farm.

11.4.5 L2/3 Trigger

The L2/3 trigger is a farm of commodity processors running Linux. After an L1 accept, all data for an event will be transferred from the Level-1 buffers to an L2/3 processor. If the event passes the L2 trigger it is then processed by L3 in the same farm node. The distinction between L2 and L3 is that L2 uses only the pixel data (or possibly pixel plus forward tracking) whereas L3 uses the full event data.

The input data to L2 consists of all L1 tracks and vertices and the raw pixel hits. The L2 algorithm performs a Kalman filter track fit on the L1 tracks and refits the primary vertices. It then searches for other primary vertices and detached secondary vertices. A secondary vertex must satisfy the following criteria: (i) tracks must have a confidence level greater than 2.5%, must be detached from the primary vertex by more than 3.5σ and must have transverse momentum > 0.5 GeV/c; (ii) all tracks must have the same sign p_z and be pointing away from the primary vertex; (iii) the vertex must have a confidence level greater than 2% and must be detached from the primary vertex by more than 3.5 σ; (iv) the vertex must have an invariant mass less than 7 GeV/c^2 and more than 100 MeV/c^2 outside the K_s mass. An event passes L2 if it has either a detached secondary vertex or a high p_T detached track. Current simulations show that we can achieve an L2 efficiency of about 90% for B decays of interest while rejecting more than 90% of light quark events.

The goal of L3 is to achieve another factor of 2 in background rejection and to reduce the size of the event to achieve a total output rate of 200 MB/s. For a 396 ns crossing time and an average of 6 interactions per bunch crossing this corresponds to an event size reduction of a factor of about three. The full offline reconstruction code will be developed and coded as part of the L3 Trigger software project. The L3 algorithm will use as much of the full offline reconstruction code as needed to achieve the required efficiency, background rejection and data reduction goals and as permitted within the latency requirements. We have prototype offline reconstruction code for forward tracking, K_s reconstruction, particle ID in the RICH, and electron, photon and π^0 reconstruction in the calorimeter.

11.5 Level 1 Trigger Hardware, Software and Simulations

11.5.1 Data Flow Analysis

We have carried out extensive data-flow modeling and simulations of trigger architectures. The data-flow modeling has provided an understanding of the design variables that affect data processing bandwidth, storage requirements, and interconnection networks. The results have been used to improve the L1 trigger architecture.

The mathematical models are based on queuing theory. Data inputs and outputs are described as stochastic processes where subsystem behavior is modeled using a set of differential-difference equations and solved for transitory states and steady states. These models provide considerable information regarding important design parameters.

The mathematical models have been validated by behavioral simulations of the L1 trigger. Input data for these simulations come from BTeVGeant, and our analysis shows that the models and simulations are in agreement.

The L1 pixel trigger includes the following hardware components: pixel preprocessors, segment trackers, network switch, and L1 Farm nodes. Figure 11.11 shows the queuing models used to model the pixel preprocessor and segment tracker hardware. For the pixel preprocessor, the Markovian model that is used to describe each stage is indicated in the figure. The segment tracker hardware is treated as a network of M/M/1 queues.

One of several data-flow analyses that has been performed has been a study of the latency introduced by the hardware that sorts pixel data. The pixel preprocessor sorts pixel data using the time stamp associated with each pixel hit. Since the amount of data varies from one beam crossing to the next and the data does not arrive in time order, we need to buffer the pixel data for a fixed period of time. We impose a 20μs latency for the sorting function. Although the latency could influence the size of the Level-1 buffers in the DAQ, we find that it does not represent a substantial contribution to the Level-1 buffer size. Detailed information on queue sizes and timing distributions for the pixel preprocessor and segment tracker hardware can be found in a separate document [14].

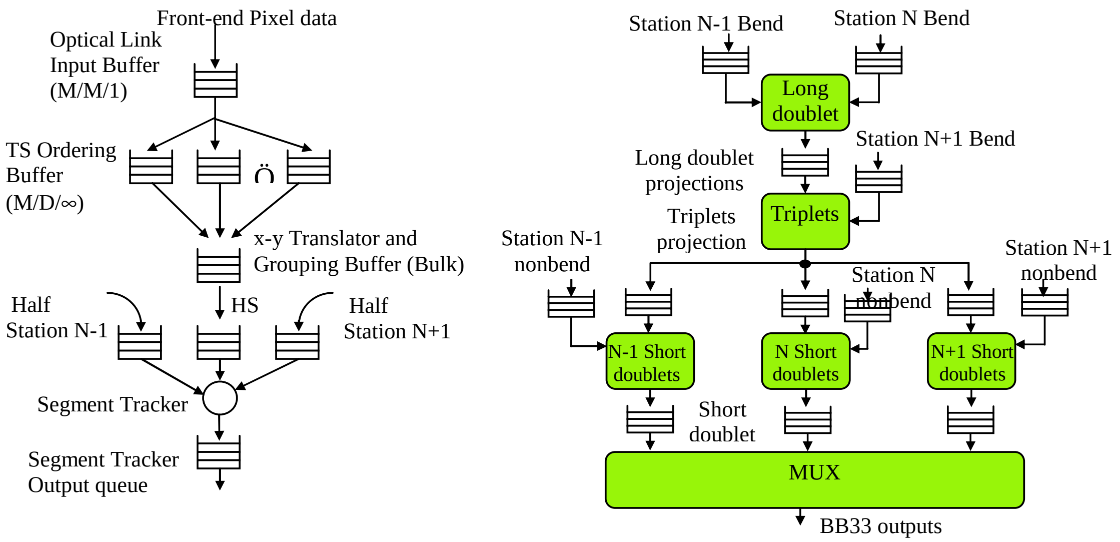

Figure 11.11: (Left) Pixel preprocessor and segment tracker queuing model, and (Right) segment tracker queuing model expanded.

11.5.2 Pixel Preprocessor

The pixel preprocessor hardware performs several functions. These functions include optical to electrical conversion, expansion of the time stamp associated with each pixel hit, sorting of pixel hits by time stamp, and hit clustering and coordinate translation. A block diagram of the pixel preprocessor is shown in Figure 11.12.

Pixel data arrive at pixel preprocessors via optical fibers. An optical receiver interface converts the high-speed serial optical data to a parallel format. The incoming data frame contains the following information:

BCO	Plane No	bend/nonbend	Chip Id.	Column	Row	ADC

The field sizes given by the FPIX chip are: BCO: 8 bits, Column: 5 bits, Row: 7 bits, ADC: 3 bits. The plane number and chip identification number are appended by the pixel detector front-end electronics. The plane number is 5 bits and the chip identification number is 7 bits. The data is packed in 16 or 32-bit words.

The time-stamp expansion function includes additional bits in the time stamp so that data can be associated with a particular beam crossing throughout the trigger and DAQ, and possibly for the lifetime of the experiment. For instance, assuming 396 ns between crossings and counting all crossings:

- 38-bits ≈ 1 day (max number≈ 2.7×10^{11}).
- 47-bits ≈ 1 year (max number≈ 1.4×10^{14}).
- 55-bits ≈ 300 years (max number≈ 3.6×10^{16})

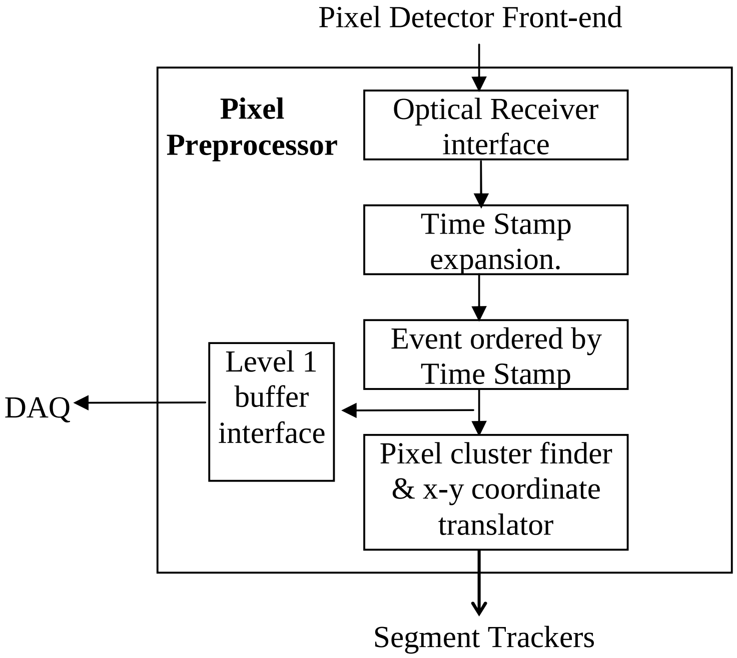

Figure 11.12: Pixel Preprocessor Schematic.

The sorting function in the pixel preprocessor hardware is critical for the trigger and DAQ. Pixel data is not strictly time ordered as it arrives at the pixel preprocessor. The sorting function sorts the data in time, and introduces a latency of approximately 20μs. The sorted data is then sent to Level-1 buffers and to the pixel-clustering hardware.

The pixel clustering and x-y coordinate translator converts pixel data to a format that is suitable for the segment tracker hardware. Due to charge sharing in the pixel detector, a track can hit more than one pixel. Hits are grouped by the pixel-clustering function, and x- and y-coordinates are assigned to each pixel cluster.

The output of the pixel preprocessor hardware is sent to three destinations, since segment trackers group hits from three adjacent pixel stations to look for track segments that consist of three pixel clusters. These track segments are referred to as *triplets*.

11.5.3 Segment Finder Hardware

This section will discuss the completed and ongoing R&D work to implement the segment finding stage of the L1 trigger algorithm in hardware. We begin by presenting a detailed description of the segment finder algorithm within the context of hardware implementation.

11.5.3.1 Segment Finding Algorithm

As described in Section 11.4.2, the segment finding algorithm looks for track segments using hit clusters from the bend view of three adjacent pixel stations which we label planes *N-1*, *N*, and *N+1*. To find inner triplets, it begins by pairing hits from *N-1* that are within an inner region close to the beam axis with all the hits from *N*. The straight line connecting the two hits is projected back to *N-2* and its $x - y$ coordinates required to be within a window roughly corresponding to the beam hole of the pixel detector.

This first stage of the inner triplet phase is also known as the long doublet finding stage. For the hardware implementation, hits from *N-1* that are within an inner region and all hits from *N* are stored in two separate lists (memory locations). The requirement that the hits on *N-1* be confined to an inner region close to the beam axis is implemented in hardware with a cut operator that filters the hits going into the list. A reverse projection operator is then applied to each pair of hits on *N-1* and *N* to extrapolate back to *N-2*. Comparators are then used to see if the extrapolated position lies within the "beam hole". As will be described below, for each hit on N-1, these operations are done simultaneously with a whole list of hits on *N* using Associative Memory (AM) or Content Addressable Memory (CAM).

All pairs passing this stage are then sent to the second stage known as the triplet finding stage. In this stage, a forward projection operator extrapolates the straight lines connecting each pair downstream to predict the upper and lower limits of the window within which a hit on *N+1* must lie. A whole list of hits on *N+1* is then simultaneously checked against each set of window limits to find the third bend view hit in the triplet. Candidates for which a hit lies within the window on *N+1* are then sent to the third and final stage known as the short doublet finding stage. In this stage, projection operators are applied to the bend view hits found in the first two stages to predict the upper and lower limits of the windows within which hits must lie in these planes. Once again, each set of window limits is simultaneously compared with a whole list of hits in each non-bend view plane. Although this stage can be carried out in parallel for each of the three non-bend view planes, it is done sequentially to simplify the hardware for assembling the hits into a triplet. Since the basic operations carried out in each of the three stages describe above are identical, their hardware implementations are also identical.

The description above also applies to the search for outer triplets except that the last stage is not implemented since matching non-bend view hits are not required.

11.5.3.2 Implementing the Segment Finder in Hardware

We have made considerable progress in implementing the segment finding algorithm in hardware. The code for this algorithm has been written in Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). Taking the behavioral model of the segment finder algorithm and implementing it in VHDL is an important step toward determining its viability. It has been proven that the algorithm can be implemented in hardware and can meet the desired performance goals.

VHDL was chosen for the implementation since it is a standard language and is widely supported by various Electronic Design Automation (EDA) tools. Though the code is written in VHDL, the design also includes the Library of Parameterized Modules (LPM) and the manufacturer's Intellectual Property (IP) functions within the source. Incorporating LPM

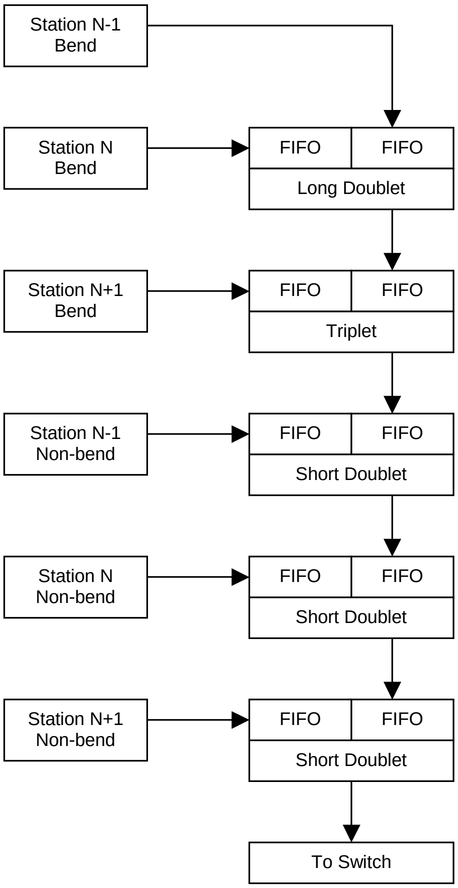

Figure 11.13: Block diagram of the segment finder algorithm as implemented in hardware.

modules increases the portability of the design between different EDA tool vendors. Using IP functions makes efficient use of each manufacturer's specific architecture and takes advantage of custom resources available within the targeted device such as embedded Random Access Memory (RAM) which is essential to this design.

A block diagram of the segment tracker firmware implementation is shown in Fig. 11.13. As described above, the segment finding algorithm processes the data in three distinct stages called the long doublet, the triplet, and the short doublet finding stages. As pixel data enters from the detector, it is stored in FIFO memory. FIFO memory is also used between each stage of the process to store results from the previous stage. Queuing both the input data as well as the intermediate results from each stage allows us to pipeline this process. When the pipeline contains sufficient data, each stage in this process will run independently of the others. A block diagram of a single stage is shown in Fig. 11.14. A general description of the process inside the long doublet, triplet, and short doublet finding stages follows.

The process starts when pixel data that meet certain constraints (e.g. hits within the

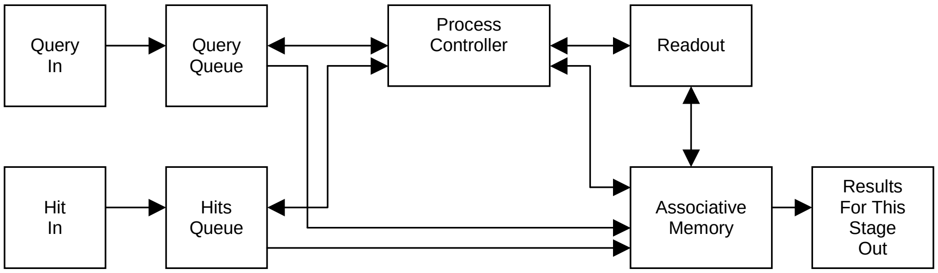

Figure 11.14: Block diagram of a single stage.

inner region of a plane) enters the Query Queue from one station. We will refer to each of these data points as a “query”. At the same time, pixel data from an adjacent station is entering the Hit Queue. We will refer to these data points as “hits”. Thus, each query will be used to investigate whether there are hits from an adjacent station that can be assembled into a segment. The values from both the Query Queue and Hit Queue are then loaded into the Associative Memory according to the processing rate of each event. The Process Controller identifies the query and matches it to the corresponding hits using the Bunch Crossing Number (BCO). Events that don’t match are discarded. In the Associative Memory, the query is stored in a single register and hits are stored in a register array that holds up to 16 values. This next step takes advantage of using an FPGA. A projection calculation is performed using the single query and all hits simultaneously in one clock cycle. Further, on the next clock cycle, all results from this calculation are simultaneously scanned and each candidate that meets the criterion is flagged for readout. Readout then assembles the results and sends them on to the next stage of the process. This same sequence is repeated in each stage of the algorithm until the complete segment is finally passed on to the switch.

The FPGA segment tracker code has been implemented using Quartus II design software from Altera [8]. A design engineer has spent 0.3 FTE over the last 3 years writing, testing, and refining this VHDL code. Throughout the design process, the segment finding code has been repeatedly run through both synthesis and place-and-route tools. The tools are used in an iterative manner to provide positive feedback on how to further optimize the code to get the best results. The results from the place-and-route tool are shown in Table 11.2. The current design uses a maximum of 80% of the available resources in an Altera EPC20K1000 part. The EPC20K1000 FPGA contains 38K total logic elements (LE’s) and belongs to a family of high-density FPGAs that range up to 114K total LE’s. This shows that the current design, together with future updates, can comfortably fit in various devices offered by Altera. Although a device from the Apex 20K family of FPGA’s has been used as the target device throughout the entire design cycle, higher speed and higher density devices now exist from both Altera and Xilinx. With some minor changes to the code, we could easily migrate this

Family	APEX20KE
Device	EPC20K1000EBC652-2X

Resources	Used	Total	Percentage
Logic Elements	30,885	38,400	80%
Device Pins	127	488	26%
Memory Bits	204,800	327,680	62%
Phase Lock Loops	1	4	25%

Table 11.2: Segment finder hardware resource usage.

design to an appropriate device in the latest generation of FPGA's. Our final objective is to target the device with the best cost versus performance characteristics regardless of the manufacturer.

Both the functional and timing simulations run on this code within the Quartus II design software use data that is extracted from Geant simulations of three adjacent pixel stations. The timing simulations have been successfully completed with the design running at 50 MHz. The results from these simulations match closely with those obtained from data flow analysis of the same data with Matlab [9].

In order to perform actual verification of the segment finding code, the design has been tested in hardware. We used a module called the Pixel Test Adapter (PTA) designed here at Fermilab as a firmware test and development platform. The PTA is a PCI based card that occupies a free slot in a personal computer (PC). The segment finding code was downloaded to an Altera EPC20K1000 FPGA on the PTA. The same pixel data files from previous software simulations were supplied to the PTA. Data is written to and results are read from the PTA card using simple C++ routines. The results from this test were consistent with those results obtained from both Matlab and Quartus II software simulations.

11.5.4 L1 Switch

Beam crossing data from the segment trackers are routed through a switching fabric to L1 worker nodes. Each of the eight highways contains one L1 Switch. For each highway, the load from the segment trackers is distributed evenly among fifteen segment-tracker nodes. The fifteen segment tracker nodes serve as input to the switch. The data at the inputs is distributed to thirty-three L1 worker nodes. Simulations for an average of six interactions per 396 ns beam crossing indicate that the output from each of the fifteen segment trackers, including excess capacity, totals 167 MB/s. The estimated total capacity needed per highway on the input side of the switch is 2.5 GB/s.

Commercial off-the-shelf Infiniband switches are capable of handling this load. Due to fixed message latencies (between 4 to 8 μs) in this type of switching fabric, small messages lead to undesirable performance. The Fermilab LQCD group has a pilot system installed

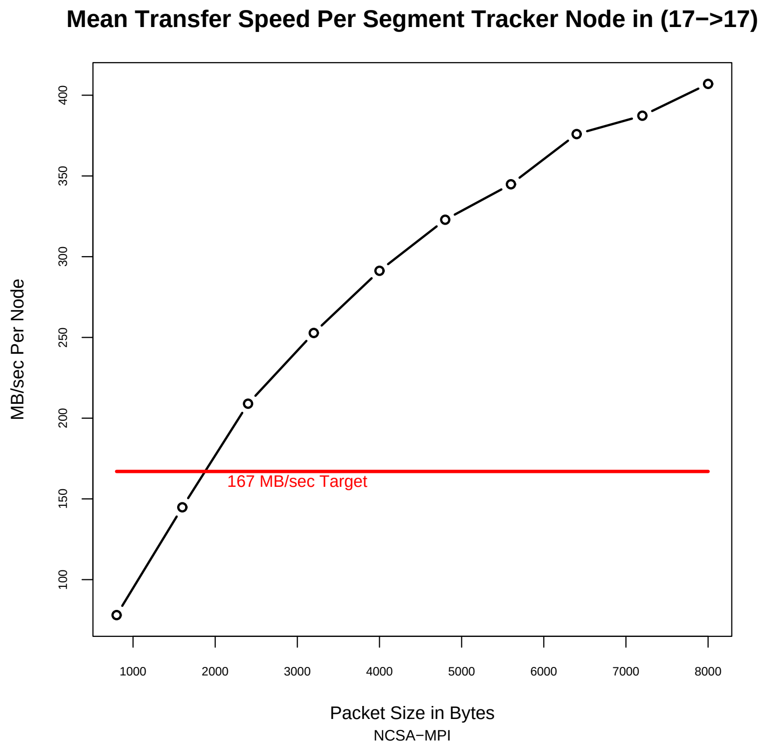

Figure 11.15: Infiniband bandwidth as a function of packet size

with 34 nodes connected to two 24-port Infiniband switches. Eight of the ports are used to interconnect the two switches. We have written simulation software to generate traffic patterns that mimic L1 beam crossing data from the BTeV detector. The software was configured to run with fixed-size messages and bandwidths were measured. There are several network libraries available for Infiniband. The simulation software uses MPICH-MPI as an application purely because it was readily available and easy to use. Drivers for the Infiniband Host Channel Adapter (HCA) are readily available for Linux/Unix from several commercial and university vendors. Almost all the work of transferring data through the switch is done by the HCA.

As the plot in Figure 11.15 shows, we exceed the required bandwidth by gathering crossing data into 2000 to 6000 byte packets at the segment tracker nodes. A 2000-byte message corresponds to seven beam crossings that are grouped into a single message at the segment trackers. A 6000-byte message corresponds to 20 beam crossings.

Additional Infiniband benchmarks can be found in references [10, 11, 12]. The NCSA group benchmarks include information about other possible switching fabrics; the main contenders are Myrinet and Gigabit Ethernet. Gigabit Ethernet latencies using standard interface cards are too high and the data rates are too low. Using smart interface cards would reduce the latency and CPU time required for data transfers.

The data transfers from segment tracker nodes to L1 worker nodes are not the only data

processor	interactions/bunch crossing		
	$\langle 2 \rangle$	$\langle 4 \rangle$	$\langle 6 \rangle$
3.2 GHz Intel P4	57.29	143.91	256.40
2.2 GHz AMD Opteron	58.11	154.65	277.70
2.4 GHz Intel Xeon	71.21	187.29	342.91
2.0 GHz IBM970PPC	81.34	210.82	378.73

Table 11.3: Poisson weighted execution times in μs for the L1 track and vertex algorithm.

paths that need to be supported. The Infiniband switching fabric has enough capacity to handle the additional load necessary to send worker-node results to Global Level-1 (GL1).

11.5.5 L1 Pixel Trigger Algorithm Timing Studies

11.5.5.1 General Purpose CPU Timing Studies

The CPU timing studies described in this section were done using a C-version of the L1 track and vertex algorithm. This code is based on the version used for the timing studies on the DSP's in the previous baseline design of the L1 track and vertex hardware [4]. A detailed discussion of the DSP timing studies can be found in Reference [6] which also describes the hash-sorter, a custom hardware implementation in an FGPA that significantly reduces the execution time of the L1 track & vertex algorithm. The pre-prototype hardware developed for the previous DSP-based design is described in detail in Reference [7].

We have measured the execution time of the L1 track and vertex reconstruction algorithm on the following general purpose processors commonly used in commercial desktop and server PC's:

- 3.2 GHz Intel Pentium 4 “Prescott” with 800 FSB (front side bus)
- 2.4 GHz Intel Xeon “Prestonia” with 400 MHz FSB
- 2.2 GHz AMD Opteron on a Sun Sunfire V20z
- 2.0 GHz IBM970 PowerPC on an Apple PowerMac G5

The L1 track/vertex code was compiled with the Intel C/C++ compiler version 8.0 on Linux for the $x86$ platforms and the IBM XL C/C++ compiler on Mac OS/X for the PowerPC platform. The execution time of the compiled code was measured using 11 simulated data files, each having a fixed number of interactions per bunch crossing ranging from 1 to 11. These execution times are shown for all three processors in Figure 11.16. The execution times for $\langle 2 \rangle$, $\langle 4 \rangle$, and $\langle 6 \rangle$ interactions per bunch crossing were then determined from a Poisson weighted sum of the results with a fixed number of interactions per bunch crossing and these results are shown in Table 11.3.

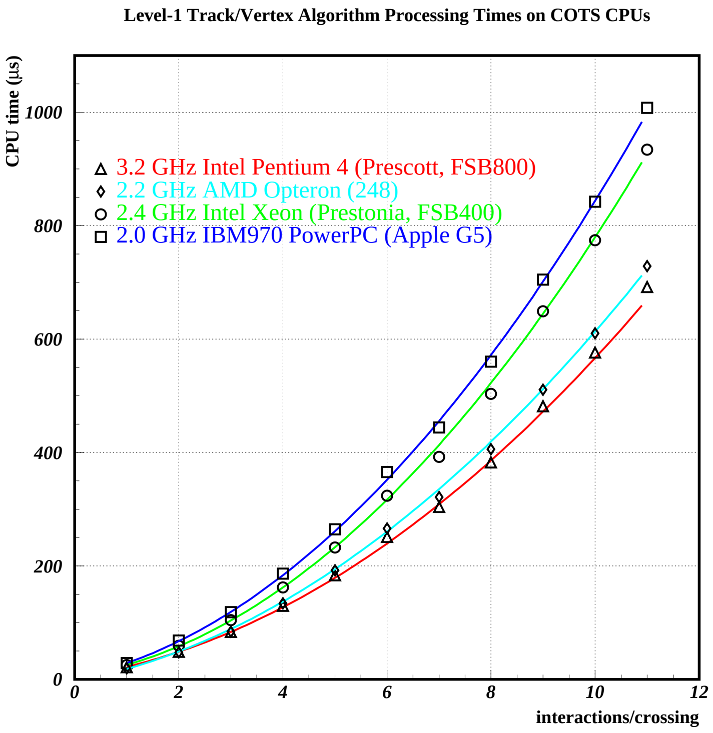

Figure 11.16: L1 track and vertex algorithm execution times on COTS processors.

11.5.6 FPGA Segment Matcher

The purpose of the tracking phase of the L1 algorithm is to identify tracks using *inner* and *outer* track segments. The segment-matcher portion of the algorithm can be broken into a small number of sequential instructions that compute many independent pairs of data. Like the segment tracker, the segment matcher can be parallelized and considered for an FPGA implementation. FPGA's are more efficient than processors for performing multiple operations with fixed-point numbers, such as comparing a single query value to a list; or adding, subtracting, or multiplying one or more values to or from a list. If the entire list fits in the FPGA memory, these operations can be done in a single clock cycle.

The L1 pixel trigger timing analysis shows that approximately 50% of the execution time for an L1 worker node is spent in the segment-matching function, and the remaining 50% is spent doing track and vertex reconstruction. If the segment matching is moved to an FPGA, then the average processing time is reduced by a factor of 2 and fewer worker nodes will be needed for the L1 farm.

The segment matching has been simulated using a behavioral model in Matlab [9] and has been implemented in VHDL. The behavioral design follows a timing model that is as close as possible to the FPGA implementation in VHDL.

Considerable progress has been made in implementing the segment matching algorithm in hardware. The experience gained developing the segment finding algorithm has made the segment matching design cycle considerably shorter. Given the similarity between the two processes, much of the VHDL code could be reused with minor modifications. The functional description of this process closely follows the description of one stage in the segment finding code. Only here, multiple projection calculations are performed using a single internal triplet with the expectation of matching only the best one of multiple external triplets stored in an array.

The FPGA segment matching code has been implemented using the Quartus II design software from Altera. This VHDL code has been compiled to target the newer generation Stratix FPGA family. In this device, the segment matching algorithm can make use of the dedicated circuitry such as multipliers, adders, and accumulators that are becoming more prevalent in the latest generation of FPGAs. The design currently uses 80% of the available resources in a Stratix EP1S30 device. Timing simulations are being performed with the design running at 70 MHz. As with segment finding, implementing the segment matching algorithm in a FPGA has enabled us to perform multiple operations in parallel and produce results using fewer clock cycles.

The option to implement the segment matcher in an FPGA gives us yet another method for reducing the number of processors needed for the L1 pixel trigger.

11.5.7 L1 Track and Vertex Hardware

The L1 track and vertex hardware will be implemented with an array of commodity CPU's and commercial network components. Based on timing studies (see Section 11.5.5.1), our baseline design consists of a total of 528 "8.0 GHz" IBM970 PowerPC CPU's (or equivalent). Compared to today's 2.0 GHz Apple PowerMac G5 Xserve, we are assuming a factor of four increase in processing power by the time we purchase the L1 worker nodes in the years 2008 and 2009. The factor of four is consistent with IBM's roadmap for future versions of the IBM970 family of processors.

Assuming a dual-CPU configuration for L1 worker nodes, as is the case for Apple's PowerMac G5 Xserve, we require 33 G5 Xserve's per trigger highway. Data from the L1 segment-tracker nodes are delivered to the L1 worker nodes via Infiniband (see Section 11.5.4). The worker nodes are managed by a system of Manager-I/O Host nodes that are connected to the worker nodes by a separate Gigabit Ethernet management network. There are two Manager-I/O Host nodes and two 24-port Gigabit Ethernet switches per highway.

An alternative to Apple's PowerMac G5 Xserve is IBM's JS20 blade server. The blade server is also based on the IBM970 PowerPC CPU, and an advantage compared to the 1U rack-mounted G5 Xserve is the higher packing density that can be achieved with blade servers. A higher packing density would reduce the number of racks needed for the L1 trigger, but this is not currently necessary since our baseline design fits comfortably into the space allotted to the L1 trigger in the C0 building. We will continue to investigate the use of blade servers for the L1 trigger, since there may be other benefits that could be exploited.

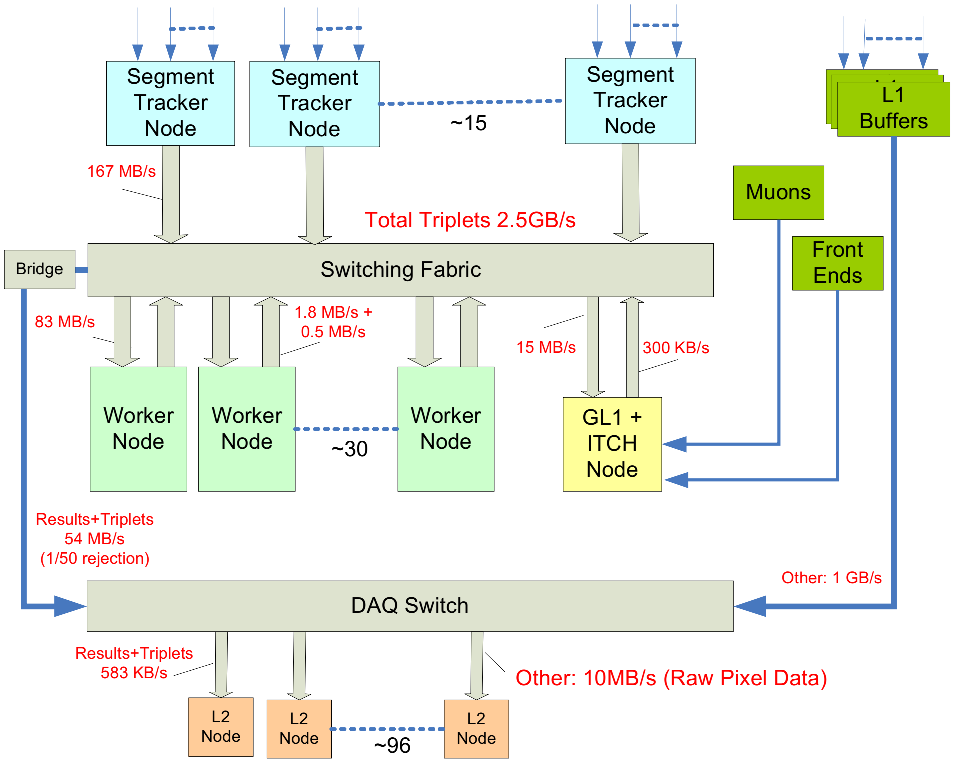

Figure 11.17: COTS based L1 trigger architecture.

11.5.8 L1 Track/Vertex Farm High-Level Software Architecture

Vital to the success of the level-1 trigger is the software running in the Level-1 farm. In order to collect data and insure its integrity, the farm must be easily configured and started, the crossing data must be efficiently transferred into physics filter algorithms, the data results must be readily available for distribution to the higher level triggers, and the entire process must be closely monitored for problems that may compromise the data. It must be possible to download new versions and configurations of the filtering algorithms. A modular design allows all these concerns to be developed as separate software components. This separation allows lower-level components to be optimized and rigorously tested independent of higher-level components.

The software infrastructure in level-1 consists of the lower-level components involved in data handling and sequencing. The infrastructure has standard, consistent interfaces to allow physics algorithms and customized monitoring code to be plugged into the system and configured. Functionality included in the infrastructure is event data network transfer, building, buffering, and dispatch, along with controls and monitoring data collection, buffering and delivery. The infrastructure will provide wrappers around vendor-provided facilities so that performance improvements can be introduced with minimal impact. The goal of good

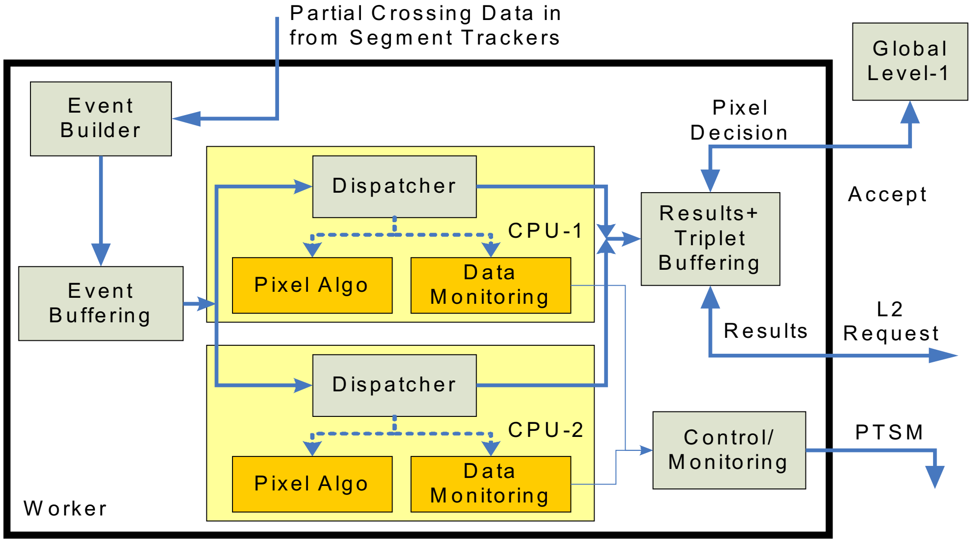

Figure 11.18: Event data flow through software components in Level 1.

infrastructure is to allow developers to concentrate on solving the problems that they are best at, and to make integration into a complex system easy and trusted. This section is mostly concerned with the organization of the software within the Level-1 worker, a block diagram of which is shown in Figure 11.18. The upstream systems are likely to be a subset or specialized version of this.

Crossing data arrives in one or more fragments. The purpose of the event builder is to collect the fragments and form entire pixel events. Once formed, the full event is placed in a queue, ready for the first of the concurrently running algorithms that is ready to grab it for processing. The dispatcher prepares the data for processing. The data is then passed in a predetermined sequence by the dispatcher through a series of algorithms and data monitors. At the end of the processing chain, the results are placed into a storage buffer waiting for a Global Level-1 accept signal and L2 requests. Sequencing performed by the dispatcher will be identically configured across all CPU's in a worker.

The physics algorithm and monitoring code are a part of a library. They are built to receive and send event data using a standard interface provided by the infrastructure. They are created independent of any data delivery systems and specific executable configuration. This feature will allow the algorithm to be used in other contexts, such as trigger simulation and release testing.

The communications between components and scheduling of the functions outlined here will be designed to maximize the time spent by a worker node in doing event processing and filtering.

Not shown in Figure 11.18 are many of the control, monitoring, and fault handling system functions. The fault handling components are covered in the RTES section of this document. The diagram shows only the data monitoring module providing data monitoring data to the outside world. An interface will be available for any component to post statistical information. The purpose of the controls/monitoring component is to make the statistical information available. It will also maintain an inventory of executables and configuration data (algorithm parameters and detector calibration and alignment). It will be responsible for preparing and running the set of event filtering programs in response to trigger start message from the operators.

11.5.9 L1 Performance

11.5.9.1 Efficiency and Rejection Studies

All studies of the L1 trigger are performed using GEANT. We generate pixel clusters, and include hadronic reinteractions, photon conversions, decays in flight, and delta rays. All studies have been performed with an average of two interactions per bunch crossing, except when we vary the number of interactions to study the trigger response for different running conditions.

We study the performance of trigger algorithms using minimum bias events and different types of B-events. We have studied a variety of cuts, implemented at various stages in the trigger, and have chosen to use a few cuts in addition to the final vertexing cuts to help reject minimum bias interactions. For example, the L1 pattern recognition eliminates low momentum tracks (tracks with $p < 3$ GeV/c) to avoid tracks that may suffer from excessive multiple scattering and could easily be reconstructed as having a large impact parameter with respect to a primary vertex. Moreover, all tracks are required to pass through at least four tracking stations to remove erroneous combinations of pixel clusters that can mimic what appear to be acceptable 3-station tracks (in these cases the interior and exterior triplets are usually constructed from identical pixel clusters). Lastly, a clean-up step removes all tracks that share pixel clusters with any other tracks. This method of removing fake tracks is simple, and perhaps overly severe, but it is effective in eliminating fake tracks at an early stage in L1. We have not performed an exhaustive study of possible trigger cuts, so we anticipate additional improvements resulting from future studies of the L1 trigger.

The L1 vertexing cuts are the final cuts that determine the L1 efficiency for B-events and the rejection of minimum bias events. These cuts are selected to provide 98% rejection for minimum bias crossings at a crossing time of 396 ns and an average of 6 interactions per bunch crossing. For a 132 ns crossing time and an average of 2 interactions per bunch crossing the cuts are selected to provide 99% rejection of minimum bias crossings. For historic reasons we describe in this section the cuts used for a crossing time of 132 ns, while the performance at 396 ns is described in a later section.

For a crossing time of 132 ns the vertexing cuts require that at least n tracks (all directed at one arm of the BTeV spectrometer) miss a primary vertex by at least $m\sigma$. The impact parameter is required to be less than 2 mm to exclude tracks that may be associated with other primary vertices in crossings with more than one interaction. The 2 mm cut also rejects daughter tracks from strange-particle decays. Fig. 11.19 shows the trigger response

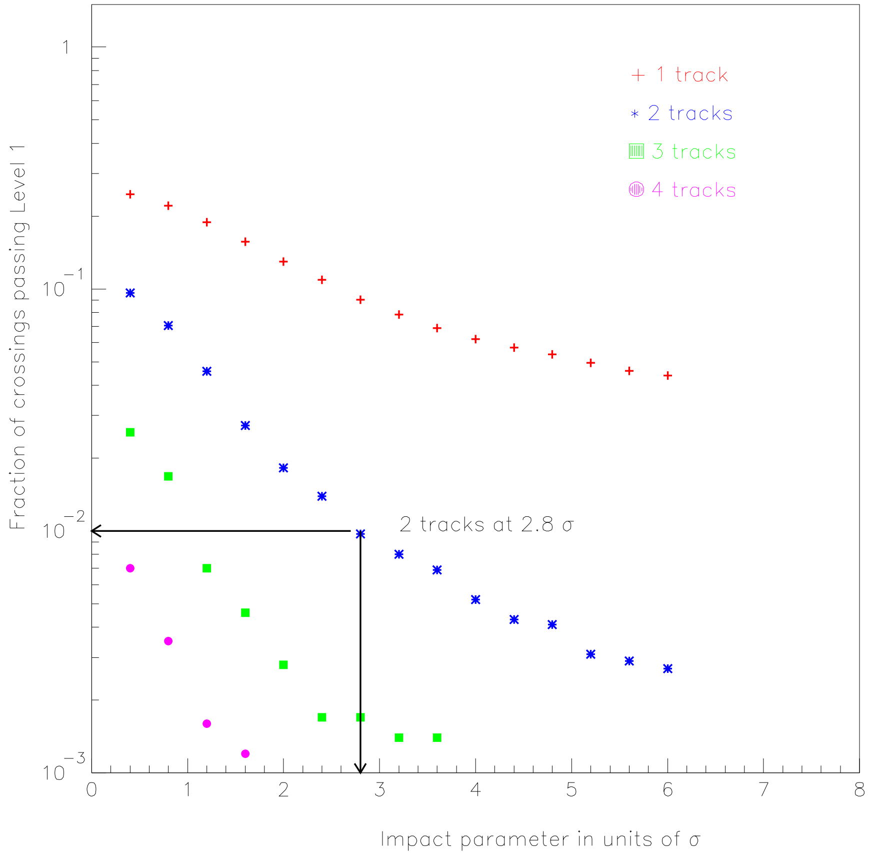

Figure 11.19: Trigger response for minimum bias events for a crossing time of 132 ns and with an average of two interactions per bunch crossing. The figure shows four sets of points requiring at least 1, 2, 3, or 4 detached tracks. The arrows show a cut that requires at least 2 detached tracks with an impact parameter that exceeds 2.8σ, and achieves 99% rejection.

for minimum bias crossings for a range of vertexing cuts. There are four sets of points corresponding to the requirement of $n = 1$, 2, 3, or 4 detached tracks. The horizontal scale specifies the minimum impact parameter for detached tracks. For the results described in this section the requirement of 2 tracks at 2.8σ is used. This provides an L1 rejection for minimum bias events of a factor of 100. For the actual experiment we would likely run with a mix of prescaled triggers.

With the vertexing cuts set to achieve the desired minimum bias rejection, we can study the trigger efficiency for different types of B-events. For $B_s \to D_s^+ K^-$ we obtain the trigger efficiencies shown in Fig. 11.20. Our cut, requiring at least 2 tracks with a minimum impact parameter of 2.8σ, gives us a trigger efficiency of 80% for this decay mode. Trigger efficiencies

Process	Efficiency	
	132 ns Crossing Time $\langle 2 \rangle$ Int/crossing	396 ns Crossing Time $\langle 6 \rangle$ Int/crossing
Minimum bias	1%	2%
$B_s \to D_s^+ K^-$	80%	66%
$B^0 \to J/\psi K_s$	65%	
$B^- \to K_s \pi^-$	45%	
$B^0 \to \phi K_s$	74%	
$B^0 \to$ 2-body modes $(\pi^+\pi^-, K^+\pi^-, K^+K^-)$	80%	

Table 11.4: L1 trigger efficiencies for minimum-bias events and various processes of interest that are required to pass off-line analysis cuts. The trigger efficiencies for all modes are determined for a bunch crossing time of 132 ns and with an average of two interactions per crossing. Results for some modes are also shown for a bunch crossing time of 396 ns and an average of 6 interactions per crossing.

for other B-decay modes are shown in Table 11.4 together with the efficiency when running at a crossing time of 396 ns and an average of 6 interactions per bunch crossing.

It is important to realize that the L1 trigger, which requires at least two detached tracks, is able to trigger on B events that involve decay modes with fewer than two charged tracks at the B-decay vertex. An example of this is the $B^- \to K_s \pi^-$ mode listed in Table 11.4. Since this mode has only one track associated with the B^- decay vertex, the majority of triggers come from detached tracks associated with the other B decay in the event.

Most of our studies of the L1 trigger efficiency are based on the decay mode $B_s \to\ D_s^+\ K^-$. These include studies of pixel noise and inefficiencies, described in the next section. Although we have not intentionally optimized cuts for this particular decay mode, it is conceivable that the current set of L1 cuts are more favorable for $B_s \to\ D_s^+\ K^-$ than for other decay modes (such as the other modes listed in Table 11.4).

11.5.9.2 Pixel Noise and Inefficiency Studies

The L1 pattern recognition is exceptionally robust with respect to pixel inefficiencies and noise hits in the vertex detector. Fig. 11.21, which shows the trigger response for $B_s \to\ D_s^+\ K^-$ and minimum bias crossings versus the number of noise hits in each pixel plane, summarizes the results from our noise and inefficiency studies. These studies were done for a crossing time of 132 ns and an average of two interactions per bunch crossing. There are three sets of points that correspond to three different pixel efficiencies. There is a noticeable decrease in the trigger efficiency for $B_s \to\ D_s^+\ K^-$ with decreasing pixel efficiency. We expect to achieve a pixel efficiency that exceeds 99%, so the trigger efficiency for $B_s \to\ D_s^+\ K^-$ at a crossing time of 132 ns and an average of two interactions per bunch

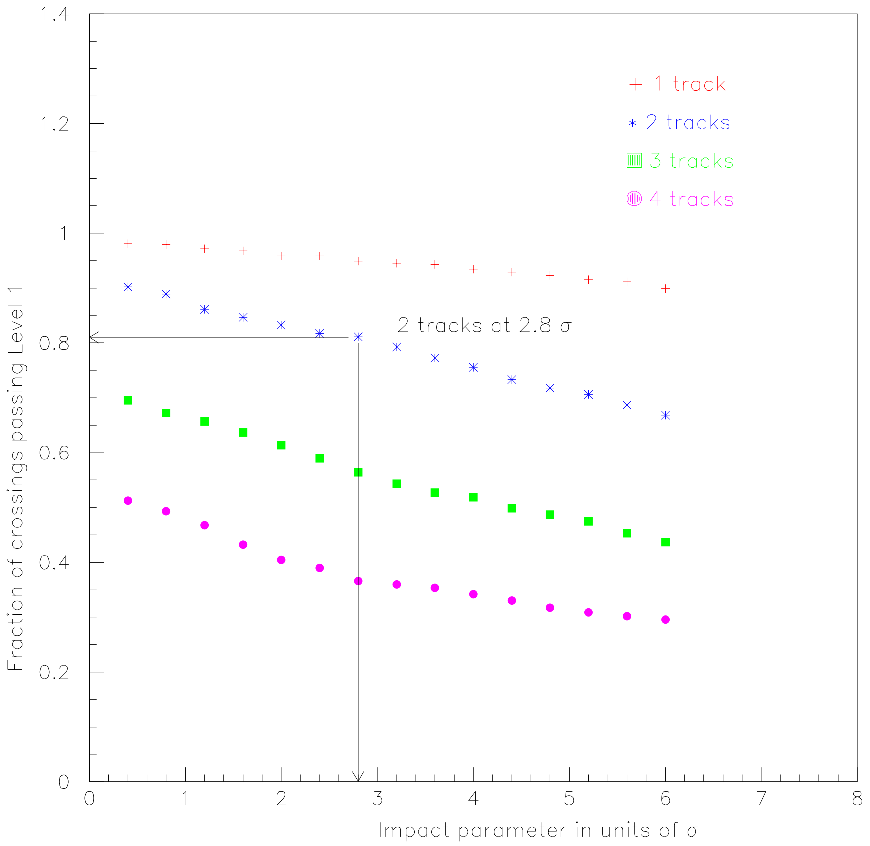

Figure 11.20: Trigger efficiency for $B_s \rightarrow D_s^+ K^-$ events with a crossing time of 132 ns and an average of two interactions per bunch crossing. The figure shows four sets of points requiring at least 1, 2, 3, or 4 detached tracks. The arrows show a cut that requires at least 2 detached tracks with an impact parameter that exceeds 2.8σ, and gives a trigger efficiency of 80%.

crossing should exceed 75%—a trigger efficiency that is less than the first set of points (stars) and greater than the third set of points (triangles).

We add noise hits to the trigger simulation to study how sensitive the pattern recognition is to spurious pixel clusters. We have studied two types of noise distributions without observing any significant difference. We generate a uniform distribution of noise over an entire pixel plane (see Fig. 11.21). We expect the noise level in the detector to be less than 10^{-5} noise hits per pixel. In our studies we observe a decrease in the trigger efficiency for $B_s \rightarrow D_s^+ K^-$ when the noise level exceeds 10^{-4} hits per pixel. This decrease probably results from the clean-up step in the L1 trigger that removes tracks with shared hits.

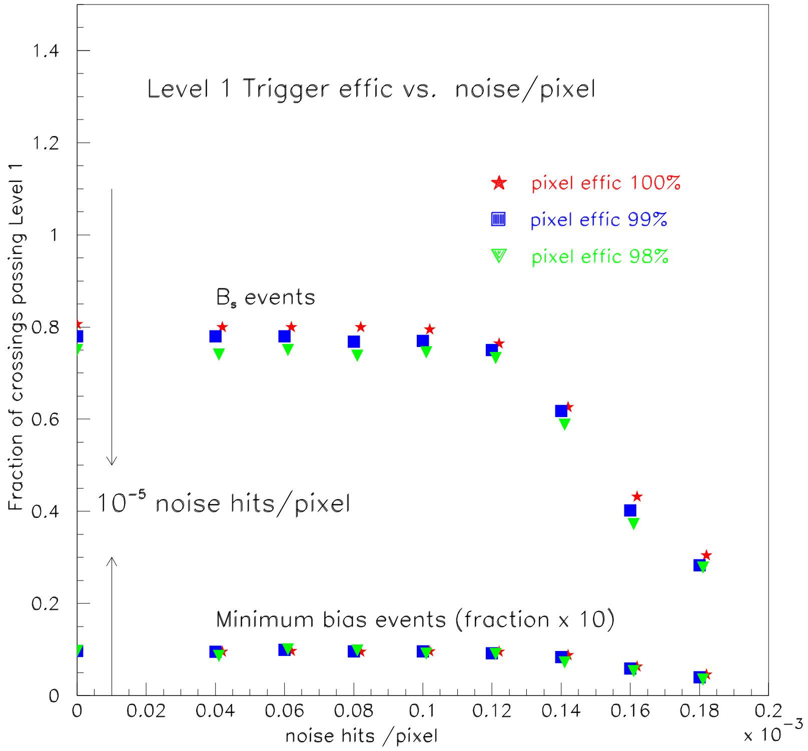

Figure 11.21: Fraction of $B_s \rightarrow \ D_s^+ \ K^-$ and minimum bias crossings, with a crossing time of 132 ns and an average of two interactions per crossing, that satisfy the L1 trigger for three different pixel efficiencies (three sets of points) vs. the number of noise hits/pixel that have been added in the vertex detector. Results for minimum bias crossings have been multiplied by 10.

For minimum bias crossings we get results that are similar to the results obtained for $B_s \rightarrow \ D_s^+ \ K^-$. Needless to say, L1 is exceptionally stable with respect to noise, and we do not expect any noticeable deterioration in the trigger performance for the amount of noise expected to occur in the pixel detector.

11.5.9.3 Performance at 396ns

This section describes studies for a range of BTeV running conditions with different numbers of interactions per bunch crossing, and the influence that these running conditions have on the BTeV trigger system.

It is important to recognize that the performance of the trigger system depends on the quality of data from the detector, the performance of the Tevatron, and even some unverified physics assumptions about the largely unexplored forward rapidity region. The trigger must be able to cope with deviations from these assumptions. It is in this context that we have studied the performance of the trigger for a range of running conditions.

For the studies described in this document we have generated samples of minimum bias interactions, and interactions with the decay $B_s \rightarrow\ D_sK$.

BTeV was originally designed to operate with a 132 ns bunch-crossing interval at an initial luminosity (at the beginning of each store) of $2\times\ 10^{32}cm^{-2}s^{-1}$. This corresponds to an average of 2 interactions per bunch crossing. The baseline running condition is now for a crossing time of 396 ns still at an initial luminosity of $2\times\ 10^{32}cm^{-2}s^{-1}$, corresponding to an average of 6 interactions per bunch crossing. The BTeV trigger system is designed to handle the range of crossing times from 132–396 ns and with 2–6 interactions per bunch crossing. We have investigated the behavior of the trigger system for different numbers of interactions per bunch crossing.

The results that we present for a range of 2-6 interactions per bunch crossing come from studies of the different hardware components used in the L1 pixel trigger, efficiency for minimum bias and B_s interactions, and bandwidth studies for data flowing into L1 and bandwidth into L2 for interactions that satisfy the L1 pixel trigger requirements. Results are summarized in a table at the end of this section.

Our studies of the FPGAs show that the time required to process pixel data increases almost linearly with the number of interactions per bunch crossing. Although the timing for this part of the L1 hardware is not an important factor (since processing times are significantly higher for the track/vertex farm), our studies do confirm that the algorithm behaves in a robust manner as the number of interactions per bunch crossing increases. Our studies indicate that the requirements for memory resources for the FPGAs also increase linearly. This is not surprising, since each additional event in a bunch crossing adds an equal amount of data.

Another study that we have performed for different numbers of interactions per bunch crossing considers the amount of data that is produced by the BTeV apparatus for each bunch crossing. This gives us an estimate of the bandwidth into L1. Furthermore, we have looked at the "size" of the interactions that are selected by the L1 trigger. There is a tendency for interactions with more data (more tracks per interaction) to be selected by L1. This determines the bandwidth into L2.

Table 11.5 shows trigger results for the L1 pixel trigger for an average of 2, 4, and 6 interactions per bunch crossing, with two different cut settings for 4 and 6 interactions per crossing. The second column shows the L1 detachment cut that we use to achieve the desired L1 efficiency for minimum-bias interactions (shown in column 5). The third column shows the bunch-crossing time, and the fourth column shows the trigger efficiency for $B_s \rightarrow\ D_sK$.

Column 6 shows our estimates of the amount of data per bunch crossing coming from the detector, and is used to determine the bandwidth into L1 (column 7). We also determine the size of events for data selected by the L1 trigger (column 8), and this gives us an estimate of the bandwidth into L2 (column 9).

For 4 interactions per crossing we show the results that we get when we use a cut that

Avg.	L 1	BCO	L1	L1	Data into L1		Data into L2	
ints/	cut(σ)	(ns)	eff	eff	kB/	GB/s	kB/	GB/s
BCO			B_s	mbias	BCO		BCO	
$< 2 >$	1.9	132	0.79	0.020	67	500	83	12.5
$< 4 >$	3.2	396	0.75	0.020	133	333	167	8.4
$< 4 >$	2.3	396	0.78	0.035	133	333	167	14.6
$< 6 >$	4.7	396	0.66	0.020	200	500	250	12.5
$< 6 >$	3.6	396	0.71	0.030	200	500	250	18.8

Table 11.5: Results are shown for an average of 2, 4, and 6 interactions per bunch crossing (BCO). The results include trigger efficiencies, timing measurements, and bandwidth determinations.

gives us the same minimum bias efficiency (2%) that we have for 2 interactions. Since the bandwidth into L2 is lowered from 13.6 GB/s to 7.6 GB/s (due to the longer bunch crossing time of 396 ns), we can relax the L1 detachment cut to maintain the bandwidth that the system has been designed for. By relaxing the cut we increase the B_s efficiency to 78%.

For 6 interactions per crossing we show what happens for two different L1 detachment cuts. First we show results for a minimum bias efficiency of 2%. Then we show results for a minimum bias efficiency of 3%, which allows us to exploit the bandwidth in our design.

We have presented results for a range of interactions per bunch crossing for the most challenging aspects of the BTeV trigger system, the L1 pixel trigger. The results show that we achieve acceptable performance for different running conditions for trigger timing and bandwidth. This shows that our baseline design for the L1 pixel trigger is able to handle these running conditions rather well, without requiring any significant changes in the design.

11.5.10 L1 Muon Trigger Hardware and Simulations

11.5.10.1 Simulation Overview

As should be clear from the discussion in Section 11.4.3, the efficiency and rejection for a given triggering scheme for the muon trigger will depend on a number of factors:

- Our choice of D-cut.

- How we choose to combine the information from the four views in each octant.

- The single-hit efficiency of the proportional tubes.

- The running conditions (i.e. non-muon occupancy).

In order to guide the design of the BTeV muon trigger and understand its performance as a function of the above factors (only some of which are under our control), we use a detailed GEANT based simulation of the BTeV detector.

In the baseline study, the minimum bias Monte Carlo events (used to study rejection) each contain N minimum bias interactions, where N is chosen from a Poisson distribution with an average $< N >= 2$. Each signal Monte Carlo event also contains N Poisson distributed minimum bias interactions, with the addition of one $B^0 \to J/\psi K_S$, where $J/\psi \to \mu^+\mu^-$ and $K_S \to \pi^+\pi^-$.

In addition to the overall fiducial acceptance defined by the BTeV muon detectors, the dimuon algorithm has two other small ”geometrical” inefficiencies.

- Since we look for tracks in all octants independently, tracks that cross between octants are usually lost.

- Since we look for a single track in each octant, $J/\psi \to \mu^+\mu^-$ events where both muons enter the same octant are usually lost.

When combined, both of the above algorithmic effects reduce the trigger efficiency for $J/\psi \to \mu^+\mu^-$ events by about 7%. This is included in the muon trigger simulation, and reflected in all efficiency and rejection numbers presented below.

In the BTeV simulation used to generate the events that feed into our study of trigger performance, the proportional tube efficiency as a function of r/R (distance from the sense wire divided by the radius of the tube) is assumed to be 100% for $0 < (r/R) < 0.85$, and 0% for $0.85 < (r/R) < 1$. Since the tubes within a view are staggered, their coverage overlaps and the probability for a muon track to hit the efficient volume of at least one tube is 100% as long as the track enters the fiducial area of that view. To study trigger performance as a function of additional proportional tube inefficiency, we use a simple random number generator in our trigger simulation code to discard Monte Carlo generated hits and achieve whatever tube hit efficiency ϵ we desire.

11.5.10.2 Dimuon Trigger Rejection and Efficiency

When calculating trigger efficiency in what follows, the denominator is the number $J/\psi \to \mu^+\mu^-$ events for which both muons passed through the fiducial area of all views in all stations (in other words, events that could have been found by a perfect trigger), and the numerator is the number of dimuon events actually found by the trigger algorithm. Rejection is defined as the total number of bunch crossings divided by the number of crossings that passed the dimuon trigger.

Fig. 11.22 shows the $J/\psi \to \mu^+\mu^-$ (i.e. dimuon) efficiency and minimum bias rejection for two trigger configurations as a function of D-cut. The configurations, each studied for two values of proportional tube hit efficiency e, are as follows:

- 2/4 views: In each octant we look for tracks in all four views (R,U,V and S). A track is defined as at least two of the four views passing the D-cut.

- 3/4 views: In each octant we look for tracks in all four views (R,U,V and S). A track is defined as at least three of the four views passing the D-cut.

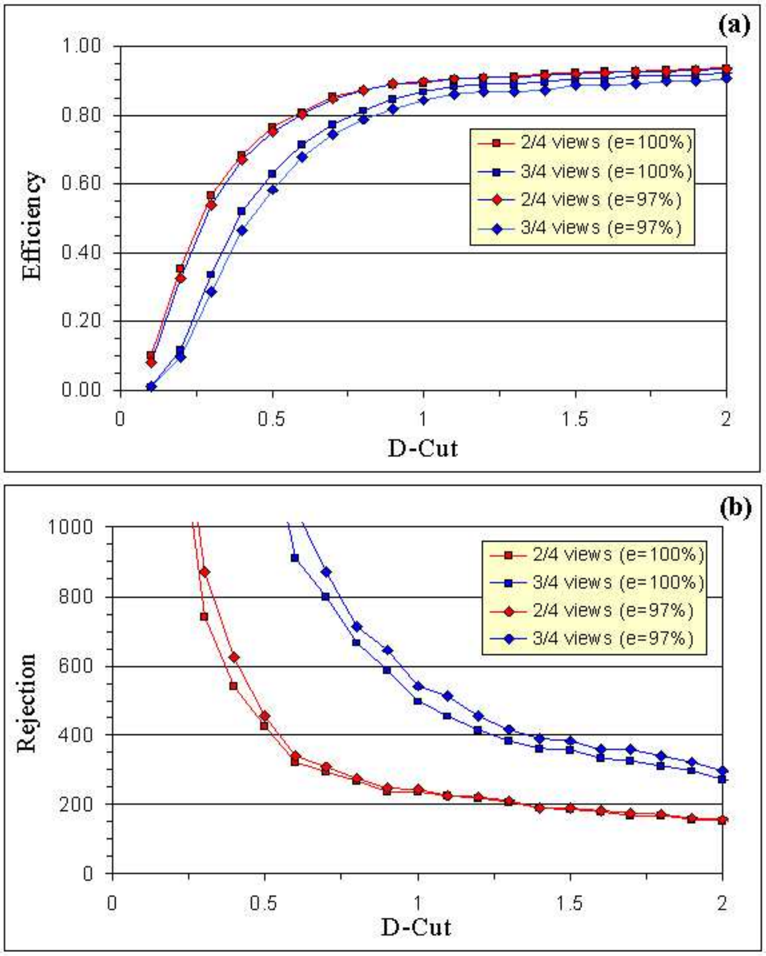

Figure 11.22: Dimuon trigger efficiency for $J/\psi \to \mu^+\mu^-$ events (a) and rejection for minimum bias bunch crossings (b), both as a function of D-cut.

Examining Fig. 11.22 we can draw some very general (and fairly obvious) conclusions:

- As we tighten the D-cut, we will drive up the rejection and drive down the efficiency for any trigger scheme.

- The looser of the two trigger schemes ("2/4") has higher efficiency and lower rejection for any value of the D-cut.

- Lowering the proportional tube efficiency (from $\epsilon = 100\%$ to $\epsilon = 97\%$ in this case) causes a relatively larger change in the tighter trigger scheme ("3/4").

- The overall effect of lowering the tube efficiency to 97% is quite small.

- The tighter "3/4" scheme has good rejection (> 500) and good efficiency (> 80%) for a range of D-cut values around 1.

These results are encouraging and indicate that we will have no problem triggering with this scheme if the assumptions that went into the simulation are valid. There is, however, some uncertainty in these assumptions. For example, although we expect it to be very high, we don't know what the proportional tube hit efficiency will be, hence a study of efficiency versus rejection for various values of ϵ is prudent.

Fig. 11.23 summarizes a study of dimuon trigger efficiency versus proportional tube hit efficiency for various triggering schemes. As above, "2/4" and "3/4" refer to the requirement that either 2 or 3 of the possible 4 views in an octant have tube combinations with $D < D$-cut. The schemes labeled "2/4(384)" and "3/4(384)" are directly comparable to the scheme used to generate the data shown in Fig. 11.22, and the others are discussed below.

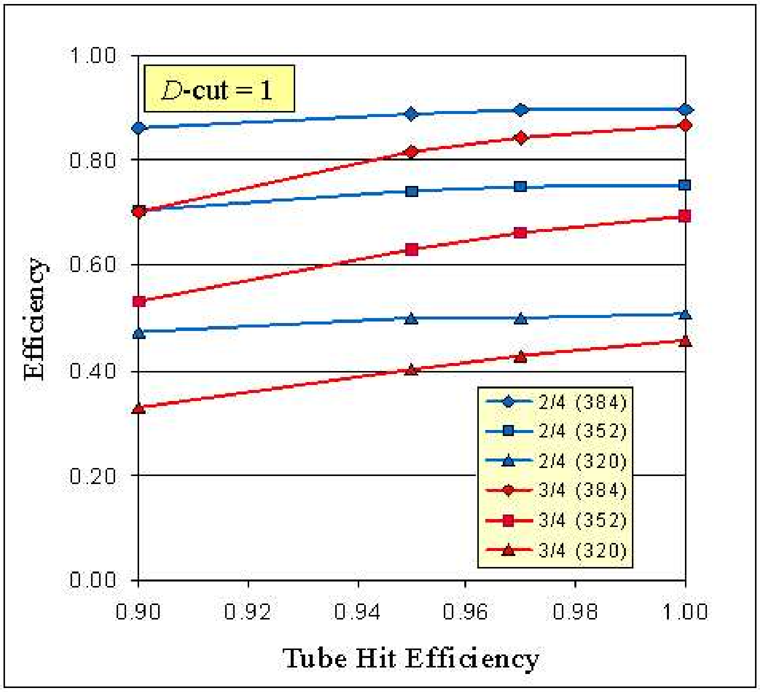

Figure 11.23: Efficiency for $J/\psi \to \mu^+\mu^-$ events for various trigger schemes as a function of proportional tube hit efficiency. A value of D-cut = 1 is used in all cases.

The numbers shown in parentheses in the legend of Fig. 11.23, "(320)", "(352)", and "(384)", indicate the maximum tube number considered in each view. To motivate the study of this additional "geometric" constraint, examine the distribution of signal (blue) and background (red) tracks in Fig. 11.6. We see that most of the background is located in the innermost tubes, closest to the beampipe. Recall that in the BTeV numbering scheme, tubes are numbered from outside in, so that these inner tubes have the biggest tube numbers.

Since there are 384 tubes in each view, the (384) classification means that all tubes are used, the (352) classification means that the innermost 32 tubes in each view are ignored, and the (320) classification means that he innermost 64 tubes in each view are ignored. Although the tactic of ignoring the innermost (i.e. hottest) tubes is not needed for the studies outlined in Fig. 11.22 and Fig. 11.23 where the average minimum bias occupancy $< N >$ is 2, we will see below that this approach might become desirable when $< N >$ is 3 or more.

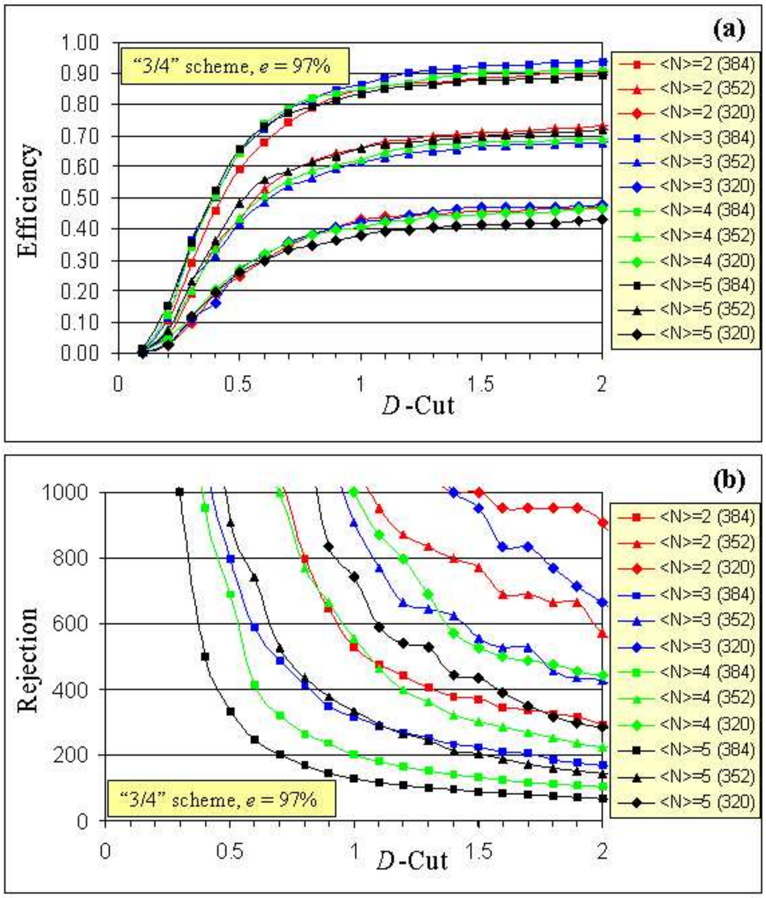

Figure 11.24: (a) Efficiency for $J/\psi \rightarrow \mu^+\mu^-$ events and (b) rejection for minimum bias bunch crossings using the "3/4" trigger scheme as a function of D-cut. Shown are the results for various average minimum bias interactions per crossing $< N >$, as well as maximum tube number requirement. The large scatter of the points at high rejection values simply reflects low statistics.

11.5.10.3 Performance at 396 ns

As suggested above, a bigger concern is the potential uncertainty in the average number of minimum bias events per crossing that we will have to deal with. If the Tevatron runs at

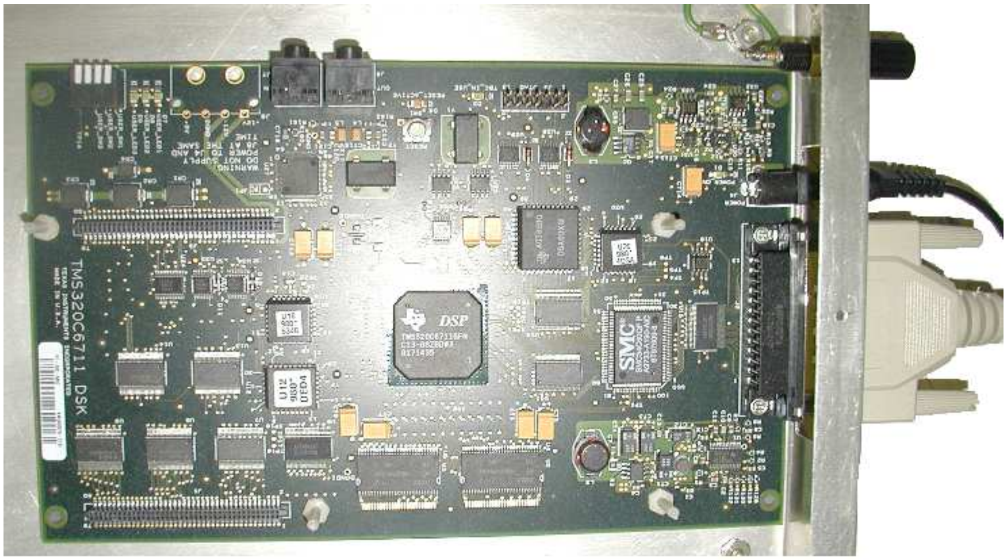

Figure 11.25: Texas Instruments TMS320C6711 DSP evaluation system.

a luminosity of 2×10^{32} and a bunch spacing of 132 ns, we expect $< N >= 2$. If we run at 396 ns bunch spacing at the same luminosity we would naively expect $< N >= 6$. For reasons we won't outline here (luminosity leveling, etc.) we expect that even if the Tevatron runs with a bunch spacing of 396 ns we may not have to deal with minimum bias occupancy beyond $< N >= 3$.

We have simulated the efficiency and rejection of the "3/4" trigger scheme for $< N > =$ 2, 3, 4, and 5, assuming a proportional tube efficiency of 97%. The results are summarized in Figure 11.24.

We see that although efficiency is largely unaffected by increasing $< N >$, the minimum-bias rejection factor falls significantly. It is still true, however, that even for $< N >= 5$ we can achieve a rejection factor of 400 with 60% efficiency. This trigger efficiency is more than adequate to satisfy the need that the dimuon trigger serve as an independent check of the efficiency of the pixel vertex trigger.

It is worth noting that significantly higher rejection as well as a much lower susceptibility to "non-muon" background hits can be achieved at the expense of efficiency by taking into account the very tight correlation between hit tubes in different views within a single station. This technique of "spacepoint" finding within some or all stations prior to correlating these hits between stations (the latter step being the approach described in detail above), was in fact the first algorithm studied in depth when developing the baseline design. While we are not using the spacepoint method in the current baseline design because of its inherently lower efficiency and slower execution speed, this can be revisited if we are faced background rates that are much worse than anticipated.

11.5.10.4 DSP Timing Test Overview

Initial muon trigger timing tests were performed using a digital-signal processor (DSP) with a real-time operating system as a model for an embedded L1 trigger processor. In an effort parallel to the muon trigger algorithm design work described in the previous paragraphs, we have done significant R&D to develop a DSP test-stand and to port a speed-optimized version of the muon trigger code to this system. Fig. 11.25 is a photograph of the Texas Instruments TMS320C6711 DSP evaluation system used in these tests. Code developed by the University of Illinois at Urbana Champaign (UIUC) group as part of the BTeV-RTES effort was used to provide an efficient file-I/O path to the board, allowing us to test the DSP based code on the same large data-sample used in the efficiency and rejection studies described above.

Additional code developed at UIUC allows us to keep track of the number of clock cycles used in each part of the test code, hence accurate speed analysis is available to further guide code optimization.

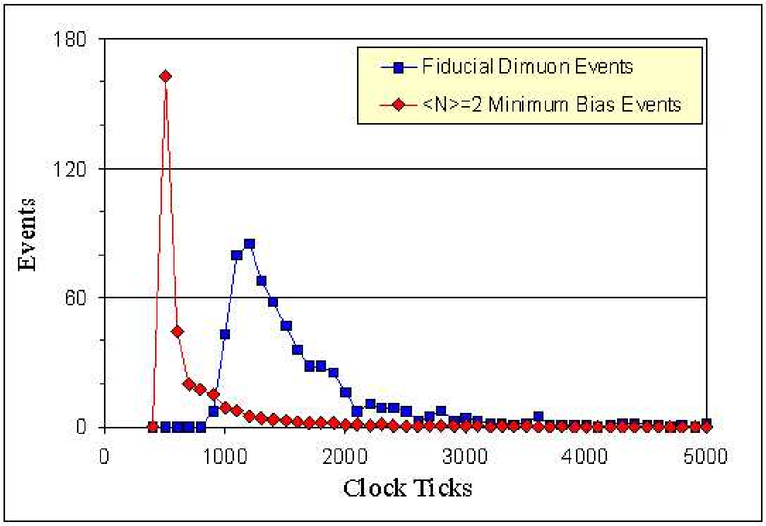

Figure 11.26: Timing results from running the muon trigger algorithm on one view in all eight octants, based on 16,000 minimum bias events (red diamonds x 1/50) and 631 fiducial $J/\psi \to \mu^+\mu^-$ events (blue squares). The average time for these distributions is 760 clock ticks and 1686 clock ticks respectively.

11.5.10.5 DSP Timing Test Results

Running the DSP based trigger code we reproduce the results discussed above, and find remarkably good speed performance. Fig. 11.26 summarized these results for samples of both $< N >= 2$ minimum bias events (red diamonds) and $J/\psi \to \mu^+\mu^-$ events for which both muons were in the fiducial volume of the muon detector (blue squares). The vertical scale for the minimum bias events has been divided by a factor of 50 to allow this large sample to appear on the same plot as the smaller and $J/\psi \rightarrow \mu^+\mu^-$ sample.

The horizontal axis is in units of CPU "clock ticks", which simply needs to be divided by the clock speed of the processor to be converted to wall clock time. For a very modest 133MHz processor, for example, the average time for the trigger code run on samples of minimum bias and fiducial and $J/\psi \rightarrow \mu^+\mu^-$ events is 6 μs and 13 μs respectively. The time shown includes running on one view in all eight detector octants. To find the time required to run on all four views, as in the "3/4" scheme discussed above, a scale factor of four should be applied.

Queuing analyses and statistical resource simulations are in progress at this time. Preliminary results confirm that using "available-today" technology (150 MHz, C6711 processors), a farm of 220 DSP's can process all 4 views (R, U, V, S), if the workload can be perfectly distributed to any free processor. However, as it is by design our plan to employ the pixel trigger farm to implement the muon trigger, there are architectural features that affect workload distribution. This is being studied, and preliminary results for timing tests are presented in the next section.

11.5.10.6 Xeon Timing Test Results

Using a 2.4 GHz Intel Xeon processor, the DSP code and dataset (described above) was re-executed. The results indicate a speed-up of a factor of 33 for the Xeon processor, compared to the 133 MHz DSP. However ,this characterizes only the processing time required for the muon trigger algorithm, and does not take into consideration the communications (I/O) bandwidth required.

Based on our DSP processing time and I/O bandwidth studies, a 10% scale factor was determined for converting pixel trigger farm resources into muon trigger resources for project cost estimates. While the timing study for the Xeon processor suggests that the L1 muon farm may require less than 10% of the L1 pixel farm hardware, the I/O considerations for the pixel and muon triggers remain unchanged. Therefore, we continue to use a 10% scale factor for the muon trigger, with the expectation that the L1 muon trigger will be I/O bound.

11.5.11 GL1 Trigger

11.5.11.1 Introduction

The Global Level 1 trigger system (GL1) accepts trigger primitives from the L1 pixel, L1 muon trigger systems and hardware triggers as defined in the trigger requirements. GL1 evaluates these primitives and generates a trigger decision for every bunch crossing. The basic elements of the GL1 system are input matching queues, GL1 algorithm processor, GL1 Supervisor and Monitor (GL1SM) and the Information Transfer Control Hardware (ITCH). Incoming data from three data streams are matched by crossing number in the matching logic or in an algorithm in the GL1 processor and then queued for the GL1 algorithm processor.

The first two streams come from the L1 pixel and L1 muon detector systems. The third stream is included for additional detector front-end electronics or control system inputs. The current anticipated use of these front-end inputs will be commissioning, diagnostic and calibration triggers. The natural extension of the trigger decision-per-crossing is the requirement that the GL1 system perform the accounting for every crossing. A timeout mechanism will be implemented to allow for lost events. Preceding subsystems are required to send a header for each crossing processed even if the data has been rejected for some reason and the header has no additional information. The incoming GL1 data will be filtered by the GL1 algorithm processor and used as an index into the current trigger table. The pre-scales are applied and a trigger decision is generated. The crossing numbers chosen for further processing are sent to the ITCH and queued to be dispatched to a L2/L3 processor over the L2/3 network. An accept message will be sent to all Level-1 buffers to cause them to reserve all the data associated with that crossing. Rejected crossing numbers will be handled by one of two procedures. Either *A)* an L1 reject message will be sent to all Level-1 buffers to cause them to free the memory space associated with that crossing or *B)* no specific reject message will be sent and the Level-1 buffer will always write over the oldest not reserved data with the newest data and keep track of what crossing numbers are currently in the buffer. The decision between these two procedures will be made based on the trade-off between the impact of many more messages on the decision network in process A and the cost of additional memory in the Level-1 buffers needed for the longest trigger latency in process *B*. The design of the memory manager to *A)* garbage collect or *B)* manage crossing pointers is also a factor here.

11.5.11.2 GL1 Input Matching Algorithm and Incoming Data

The baseline architecture supports incoming event packets of 50 to 100 bytes from the L1 pixel and L1 muon trigger systems. All events will have the BTeV standard bunch crossing number in the header and all inputs will be matched according to this number. It is required that auxiliary front-end trigger inputs arrive at GL1 at the same time or before the arrival of the matching muon or pixel event packet. The exact time will be determined by the production hardware, but it will most likely be 1μs or less. The matching algorithm will collect information by crossing number until it has all three input packets (or two if the auxiliary inputs are disabled) or until a timeout occurs. The GL1 system needs to have the same accelerator based clock and timing inputs as the front-end DCB's so that GL1 is synchronized with the front-ends. The matching algorithm will then queue the data for the GL1 processor(s) which will process the information in the order of arrival.

11.5.11.3 GL1 Algorithm Processor

The GL1 processor will most likely be a small collection of processing elements because of the volume of information and data rate requirements. The matched packets from the three data streams will be filtered for already rejected events, minimum data and then combined by the trigger algorithm and indexed to the trigger table to produce an intermediate result.

That result will be indexed through a prescale table to produce an ultimate crossing decision, that is, a go/no-go decision on whether the L2/L3 farm should process this crossing of data. The decisions will be classified by type as defined in the trigger algorithm in order to allow simple optimizations later. The GL1 trigger decisions might be ordered by crossing number if it is advantageous for subsequent processing. For instance, the L1 buffers would have a less complicated algorithm if they could infer that earlier beam crossings could be rejected after receiving an L1 accept for a particular beam crossing. The drawback to this approach is that the ordering of GL1 decision messages means that trigger decisions will be delayed while waiting for the processing to be completed for beam crossings that need more time. Moreover, the GL1 decisions are queued into the ITCH, which would also need to respect any constraints on time ordering. The GL1 processor may also produce and store some additional information that could help later processing stages. This may include portions of the incoming data that are not already saved in Level-1 buffers, intermediate results and/or trigger primitives to jump-start later processing. This information can be added to the GL1 Level-1 buffer or simply passed through the ITCH to make it accessible to L2/3.

11.5.11.4 Information Transfer Control Hardware (ITCH)

The ITCH must match the selected trigger data that needs further processing with resources that may be limited in availability. The processing resources are in the L2/3 farm and each processor may be powered off, doing diagnostics, be ready to process with an algorithm limited in some way or be available for any trigger type. Those processors register with the ITCH as their processing resources become available. The registration mechanism allows for the L2/3 processor to indicate a desired (or required) trigger type. The ITCH maintains queues of crossings to be processed and resources available, finds matches between the two and sends to the available processing element the crossing number to work on. It may also notify all of the Level-1 buffers of the assignment of data to resource although an alternative being considered is to have the resource request the data itself. That mechanism needs simulation to determine the trade off between latency and additional load on the network, similar to the accept/reject evaluation above. If a processing element has considerable capability, it can continue to request crossings until it reaches an acceptable load. The ITCH must monitor the data queue sizes, report unacceptable levels and request resource additions, L2/3SM, and/or Run Control. The ITCH is also the trigger accountant, logging missing crossings and collecting the information needed to monitor trigger operation and refine the trigger tables. The ITCH may communicate with GL1 to adaptively swap trigger tables to optimize the trigger system to match the luminosity of the accelerator, to adapt to changing levels of processing resources or to accommodate to changes in other detector systems.

11.5.11.5 GL1 Supervisor and Monitoring (GL1SM)

The GL1SM will oversee the GL1 processing elements, distribute commands from the BTeV DAQ/Run Control system, assist with error mitigation, collect and report physics performance and hardware diagnostic information. Some of this information can be displayed as histograms for the trigger operator and/or the experiment shift team. There will be a trigger expert terminal and display connected directly to the GL1SM, allowing a privileged operator to monitor, diagnose and adjust the GL1 trigger system independent of the rest of the experiment systems. More details on the GL1SM can be found in Section 11.7.

11.6 L2/3 Hardware, Software and Simulations

11.6.1 L2/3 Hardware

11.6.1.1 Overview of Baseline Design

The L2/3 trigger will be implemented with an array of commodity CPU's and commercial network components. Fig. 11.27 shows an illustration of the layout of the components of the L2/3 trigger. The 1536 CPU's in the baseline design consist of 768 1U rack mount dual-CPU PC's running Fermilab Linux. These are split equally between the 8 DAQ highways, 96 PC's (192 CPU's) per highway. Data from the L1 Trigger and the Front Ends are delivered, by way of the L1 buffers, to the L2/3 PC's via DAQ Fanout switches.

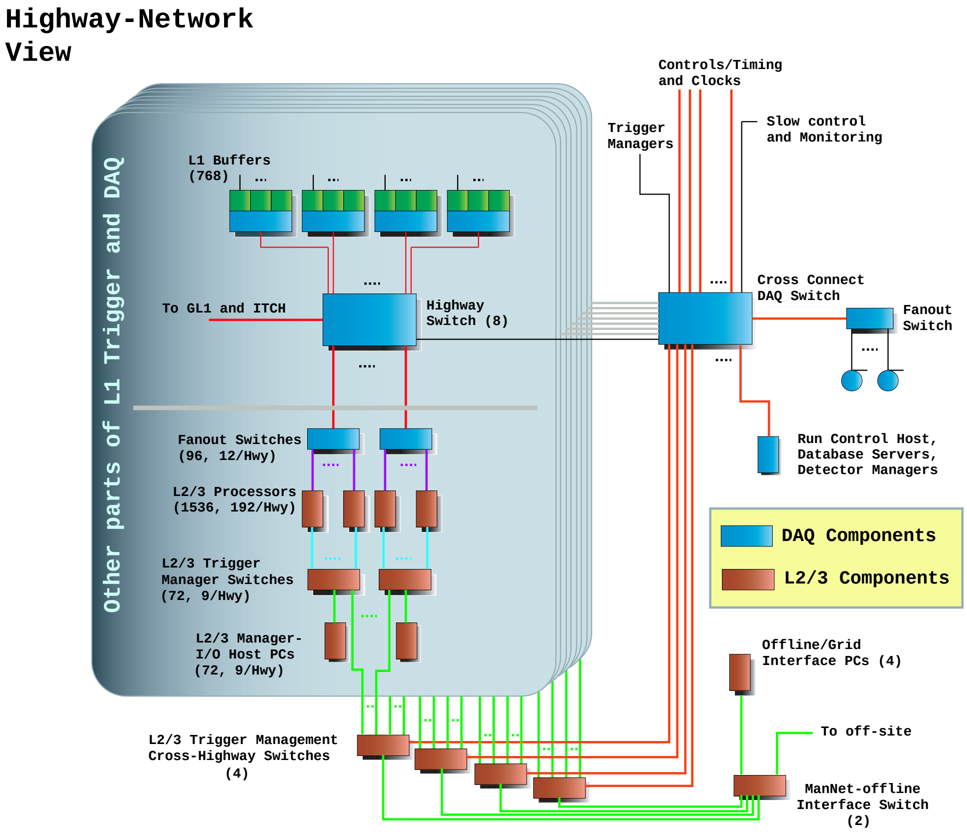
Highway-Network View
Other parts of L1 Trigger and DAQ
L1 Buffers (768)
To GL1 and ITCH
Highway Switch (8)
Fanout Switches (96, 12/Hwy)
L2/3 Processors (1536, 192/Hwy)
L2/3 Trigger Manager Switches (72, 9/Hwy)
L2/3 Manager-I/O Host PCs (72, 9/Hwy)
Controls/Timing and Clocks
Trigger Managers
Slow control and Monitoring
Cross Connect DAQ Switch
Fanout Switch
Run Control Host, Database Servers, Detector Managers
DAQ Components
L2/3 Components
Offline/Grid Interface PCs (4)
To off-site
L2/3 Trigger Management Cross-Highway Switches (4)
ManNet-offline Interface Switch (2)

Figure 11.27: Illustration of the layout of the L2/3 Trigger.

The L2/3 Trigger CPU's are managed by a system of Manager-I/O Host PC's connected via a separate management network. This management system consists of 9 Manager PC's and 9 (Manager) switches per Highway. The Manager PC's from one Highway can communicate with the Manager PC's from other Highways via a Management Network (ManNet) Cross Connect switch. The Manager PC's are used for managing and control of the L2/3 farm worker PC's, as servers for database caches and output event pool caches as well as for use in monitoring the farm worker PC's, the data networks, and other components of the L2/3 Trigger system.

The L2/3 Trigger system will be located in a total of 42 racks on the third floor of the counting room, two floors above where the hardware for the L1 Trigger will reside. The 96 farm worker PC's for each Highway will be housed in 4 racks, equally split between each rack. Each of these 32 racks will house the necessary DAQ Fanout and Trigger Manager switches required for the PC workers in that rack. Each farm worker PC will have two copper gigabit ethernet (1000Base-T) connections to two of the DAQ Fanout switches. The manager PC system will be housed in 8 additional racks while 2 racks will house the other L2/3 specific hardware.

Technologies using small single board computers running embedded Linux were investigated but these do not provide enough processing power. The relatively new blade servers are being investigated as a possible alternative to 1U dual processor PC's. Currently their cost per processing power is still too high. However they may become a viable alternative in the future. The advantages of the blade server are higher density and potentially lower power needs and better reliability and manageability. If enough power and cooling can be placed into each rack, with blade servers the 1536 farm worker CPU's could potentially be house in just 6–10 racks instead of 32 racks. In the future we expect the blade servers to become almost as much a commodity item as the 1U PC's are now, due to the current trend of processor improvement and mass market interest in blade servers. The L2/3 Trigger processing farm is expected to be heterogeneous.

Network Data Rate

In the baseline design the processing for the L3 Trigger takes place on the same PC as the processing for the L2 Trigger. The difference is that the L2 Trigger uses only a subset of the total available information, (the pixel tracking), while the L3 Trigger uses the entire event information and performs as much of the event reconstruction as needed to satisfy the efficiency, background rejection, and event data reduction goals within the allowable processing time.

The rate of events (crossings) entering the L2 Trigger is 50 KHz. With an average event size of 250 KB/event, this means the average data rate entering L2 is 12.5 GB/s, or 1563 MB/s for each DAQ Highway. Each Highway has 32 8-port DAQ Fanout switches, this means that each 8-port DAQ Fanout switch must handle a average data rate of about 49 MB/s, or 391 Mbps. This can easily be achieved with two Gigabit connections between the Highway Switch and each of the DAQ Fanout switches.

Each Fanout switch distributes data to 3 Farm worker PC's, giving about 16 MB/s per PC, or 8.1 MB/s per CPU. This corresponds to 130 Mbps per PC, or 65 Mbps per CPU. In the baseline design this is achieved with two copper gigabit ethernet (1000Base-T, 1000 Mbps) connections between each PC and Fanout switch.

The Manager PC's communicate with the Farm Worker PC's via a separate management network. This network consists of 24-port 10/100 Manager switches with two Gigabit ports included. The Gigabit connection is from the Manager PC to the Manager switches, while single fast ethernet connections are used to connect the Farm Worker PC's to the Manager Switches. A separate management network helps achieve reliability and fault tolerance, as well as provide network resources for offline processing needs.

In the baseline design, after a L1 Trigger accept, all data for the event will be transferred to a L2/3 Trigger Farm Worker PC.[1] If an event passes the L2 Trigger, it is processed in the same PC through the L3 Trigger. Hence there is no data transfer through the network for entry into the L3 Trigger. The maximum average output rate of the L3 Trigger is 200 MB/s total. At 396 ns crossing time this corresponds to a rate of 2.5 KHz with an average event size of 80 KB. This translates into 130 KB/s per CPU or 1.0 Mbps per CPU. This is small compared to the incoming data rate and can be handled by the baseline DAQ network design without additional hardware on the Farm Worker PC's. The output data rate is 25 MB/s for each Highway, where the data is transferred from the Farm Worker PC's to the DAQ Highway Switch via the DAQ Fanout switches. From the Highway Switch the output goes out to data logging machines via the DAQ Cross Connect Switch at 200 MB/s.

CPU Speed

The L2 Trigger takes an input rate of 50 KHz and reduces it by a factor of 10 to 5 KHz. The L3 trigger further reduces this rate by another factor of about 2 to 2.5 KHz (as well as reducing the event size by a factor of 3.) This can be achieved with 922 CPUs if the L2 Trigger processing takes less than 5 ms/event per CPU and the L3 trigger takes less than 134 ms/event per CPU. These latency times represent 100% CPU utilization which is not achievable. In the baseline design 1536 CPUs will be used in the L2/3 farm which corresponds to a CPU utilization of 60%. Note that a maximum of 10% of the CPU is allocated for DAQ event building and another 10% is allocated for RTES related and other monitoring. We assume that the overheads of the OS and of switching between tasks is 10%. The remaining 10% CPU time is considered as extra headroom.

To estimate the likely performance of a CPU that would be used in the BTeV L2/3 Trigger Farm we looked at the CPU core speed trends in the last ten years. The CPU core speeds for INTEL and AMD CPU's is plotted against the release date of the CPU is given in Fig. 11.28. For later AMD CPU's where AMD quotes a performance rating (relative to a P4 CPU), this rating is used to provide alternative data points. Also shown on the plot are the Moore's Law curves for a doubling of CPU speed every 18 and 24 months. It can

[1] Note that in order to make efficient use of the network, in fact a buffer of many events rather than a single one will be sent to each PC for each data transfer request.

be seen that the data shows that the CPU speed doubling time to be between 18 or 24 months. The L2/3 farm CPU's will be bought over 1.5 years, 25% in FY07, 25% in FY08 and 50% in FY09. Using these fractions we can calculate an average CPU speed for a L2/3 Farm CPU. For CPU speed doubling times of 18, 24 and 30 months, the average L2/3 Farm CPU speed is projected to be 23 GHz, 14 GHz and 10 GHz respectively. For the baseline design requirements on the L2 and L3 trigger processing latencies, we have chosen that all processing times be compared against a CPU that will run the code 4 times faster than on a 3 GHz P4 Xeon CPU. We call this a "12 GHz P4 CPU" for the purposes of this document.[2]

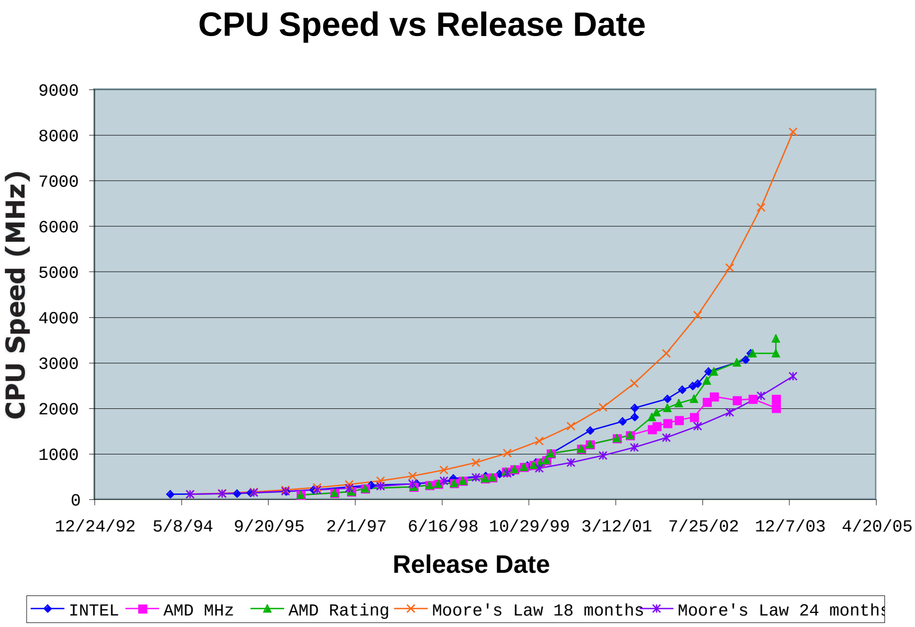

Figure 11.28: CPU core speeds plotted against the release date of the CPU.

The L2 Trigger code which achieves the required rejection was run on various PC's with different CPUs under Fermilab Linux. The results of this study are shown in Table 11.6. The data was generated using BTeVGeant. Both minimum-bias and $b\bar{b}$ events were run. It can be seen that we can satisfy the latency requirement using existing CPUs like those listed in Table 11.6. The original challenge was to achieve the desired L2 Trigger efficiency and rejection with a CPU of reasonable speed. After work on improving the L2 algorithm, this has been shown to be achievable even with existing, low-end CPUs. We therefore expect that the CPU requirements will be set by the L3 Trigger code.

[2] Recent news from CPU manufacturers indicate that the highest projected CPU core speeds may not be reachable with the 0.090 μm process CPUs, and dual-core CPUs are at this time anticipated. For L2/3 Trigger processing and for the purposes of this document, a dual-core "6 GHz P4 CPU" would be equivalent to a "12 GHz P4 CPU", since we would simply double the number of running L2/3 filter programs within a PC.

CPU Type	**CPU Speed**	**Time/Event (ms)**					
	(GHz)	**2 Int/Cross**		**4 Int/Cross**		**6 Int/Cross**	
		M.bias	$\boldsymbol{b\bar{b}}$	**M. bias**	$\boldsymbol{b\bar{b}}$	**M. bias**	$\boldsymbol{b\bar{b}}$
INTEL P3 (SLOT 1)	0.5	9.5					
INTEL P3 (Coppermine)	1.0	4.3	5.2				
INTEL P3 (Coppermine)	0.866	5.0					
INTEL P4 (Xeon)	2.4	2.3		2.9		3.2	
AMD ATHLON	1.2	3.0	3.2	3.8	4.1	4.3	4.5
"12 GHz P4" (Projected)	12	0.46	0.49	0.58	0.63	0.64	0.67

Table 11.6: The times taken for execution of the L2 Trigger code on various CPUs for various numbers of interactions per crossing, for both minimum bias only events (crossings) and those crossings also containing $b\bar{b}$ decays.

Besides the CPU speed and the actual Trigger code, the version of the compiler used and the optimization selected can also affect the execution speed. The L2 code was written in C++ and the compiler used was GNU gcc version 2.95.2. We expect that the compiler will improve in the future which will produce faster executing code.

There is currently no full L3 trigger code to benchmark. Although we have done many physics analyses of simulated data that includes reconstruction of neutrals and particle ID, we are not at the stage of having a full reconstruction package, unlike in a mature running experiment. However the additional rejection of a factor of 2 in the trigger rate is relatively modest and we show that the latency of 134 ms/event per CPU should be achievable with a CPU equivalent in speed to a 12 GHz Pentium IV Xeon CPU.

First we obtain the approximate time it takes to do the L3 tracking. We have already benchmarked the L2 code and found that pixel-only tracking (for tracks associated with the L1 trigger vertex) takes less than 5 ms on a 2.4 GHz Pentium 4 CPU. To get an estimate of the L3 trigger using also the forward tracking detectors, a preliminary version of the L3 tracking was benchmarked that performed the full tracking including hits found in the forward tracking detectors, for all tracks found at L1. The execution times for this preliminary version of the full tracking is shown in Table 11.7. The projection to a "P4 12 GHz CPU" is done using a linear fit of the inverse of the processing time versus the CPU core speed. This is shown in Fig. 11.29, where for the AMD CPU we have taken the Pentium equivalent CPU core speed which is about 30% higher than the actual physical clock rate. We are confident that from these results the charged particle tracking and therefore the minimum required L3 Trigger code can be run comfortably within the required maximum latency of 134 ms.

L3 Trigger and Offline Processing

The goal of the L3 Trigger is to achieve another factor of 2 in background rejection and to reduce the size of the event to achieve a total output rate of 200 MB/s. For a 396 ns crossing time and an average of 6 interactions per bunch crossing this corresponds to an event size

CPU	**Time/Event (msec) for $b\bar{b}$**	
	2 Int/crossing	**6 Int/crossing**
Pentium III 866 MHz	201	433
Pentium IV Xeon 2.4 GHz	94	208
Pentium IV 3.2 GHz	73	153
AMD Athlon 1.2 GHz	135	298
AMD Athlon 2800+ 2.1 GHz	79	170
"P4 12 GHz" (projected)	22	45

Table 11.7: Execution or projected execution times for a preliminary version of the L3 charged particle tracking as described in the text. (The projection is based on a linear fit of the inverse of the processing time versus the CPU core speed.)

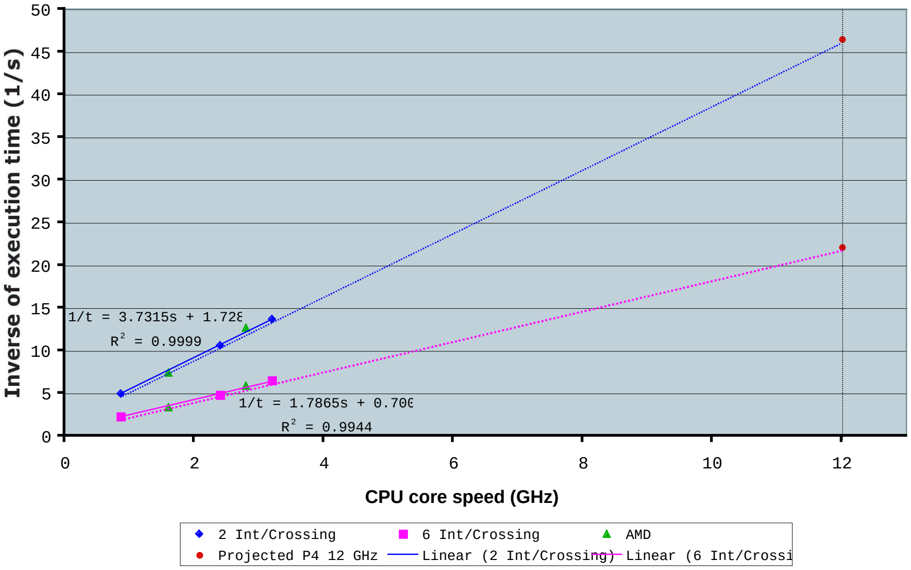

Figure 11.29: Inverse of the preliminary L3 tracking processing times versus the CPU core speeds to project the processing times for a "P4 12 GHz CPU". (The CPU core speed used for the AMD processors are the equivalent P4 speeds.)

reduction of a factor of about three. The full offline reconstruction code will be developed and coded as part of the L3 Trigger software project. The L3 algorithm will use as much of the full offline reconstruction code as needed to achieve the required efficiency, background rejection and data reduction goals and as permitted within the latency requirements. Although it may be desirable to run the full (offline) reconstruction in the L3 Trigger if possible, this is very likely not necessary to achieve the stated L3 Trigger goals. Additional offline processing can be done on the L2/3 PC farm during the substantial idle periods (we assume a duty cycle of 33% for beam) or at the PC farms of collaborating institutions.

We already mentioned above that the charged particle tracking and therefore the minimum required L3 Trigger code can most likely be run comfortably within the maximum allowed processing latency. This will meet the background rejection requirement, but an uncertainty exists in the level of data reduction that can be achieved. This is addressed in the next section using only part of the full reconstruction. Since the data reduction is made easier if we can do the full event reconstruction at the L3 Trigger stage, in this section we try to estimate the likely processing time for the full event reconstruction if it were run at the L3 Trigger stage.

To estimate the amount of processing time required for the full event reconstruction we used the reconstruction code from FOCUS, a fixed-target charm photoproduction experiment that ran in 1996-7, and from E791, a fixed-target charm hadroproduction experiment that ran in 1990-1991. We used these as a benchmark of how long the full event reconstruction code could take, as well as looking at the CDF and D0 RunII reconstruction experience. Unlike for the real BTeV L3 code, these offline reconstruction code were not highly optimized for execution speed, however it should give an idea of the likely execution time. Since the interactions and the spectrometer of BTeV is more complicated than that of FOCUS or E791, we must renormalize the FOCUS and E791 timings by the average number of primary vertices produced, by the number of secondary particles produced and by the number of detector channels. This is done in Tables 11.8 and 11.9. The average of the projected execution times for FOCUS and E791 code is 134 ms on a "12 GHz P4 CPU". Note that the FOCUS and E791 offline reconstruction code was not optimized for execution speed as trigger code would be. We expect significantly faster executing reconstruction code when attention is paid to speed.

As another comparison we also look at the CDF and D0 RunII experiments which run at the Tevatron but with a central detector. Since we would like the BTeV L3 Trigger to do as much of the traditionally offline reconstruction, we compare to the CDF and D0 offline (as opposed to their trigger code), see Fig. 11.30. For data taken at an average luminosity of 2.0×10^{31} cm^{-2}s^{-1}, the D0 offline processing takes about 25 sec/event on a "1 GHz CPU" This is much higher than their original specifications due to slow code not yet optimized for speed and due to software developers' appetite for including additional processing. We expect they could gain significant improvements in code speed. The CDF offline code is more tied with their L3 trigger code and thus may represent a better comparison to maybe show the gain in code execution speed when one is coding with an online trigger in mind rather than offline. The CDF offline code takes about 2.5 sec/event on a "1 GHz CPU" for

Description	**Time/event (ms)**
1 million events in 8 hours (SGI Indy R5000 150MHz)	29
normalize to multiplicity at primary $\times\frac{17}{4}$	123
normalize to number of primary vertices $\times\frac{9}{2}$	554
normalize to number of detector channels/segment $\times 10$ (BTeV/FOCUS: SSD 128/8; straws/PWC 27/13; ECAL 11/3; pixels in L2)	5540
normalize to 500 MHz Pentium III CPU (using g77)	2928
normalize to 1.0 GHz P4 CPU	1464
normalize to "12 GHz CPU"	122

Table 11.8: Execution or projected execution times for FOCUS code. Renormalizations are made to try to project to a BTeV-like event. Each normalization is cumulative.

Description	**Time/event (ms)**
Run on an SGI Indy R4000 100MHz	160
normalize to multiplicity at primary $\times\frac{17}{7}$	389
normalize to number of primary vertices $\times\frac{9}{2}$	1749
normalize to number of detector channels/segment $\times 10$	17490
normalize to 1.0 GHz P4 CPU (using Tiny)	1749
normalize to "12 GHz CPU"	146

Table 11.9: Execution or projected execution times for E791 code. Renormalizations are made to try to project to a BTeV-like event. Each normalization is cumulative.

data taken at an average luminosity of 2.0×10^{31} cm^{-2}s^{-1}. This is ten times faster than the D0 code. Part of this difference is due to the quite different numbers of tracking elements in the two detectors, but we believe part of this is also due to more attention being paid to execution speed. (At an average luminosity of 2.0×10^{32} cm^{-2}s^{-1} we project the CDF code to take 658 ms on a "12 GHz P4 CPU"). Thus although the CDF and D0 offline code runs much slower than our goal of 134 ms/event for the BTeV L3 Trigger, this difference can be substantially reduced when proper care and consideration of code speed is taken into account right from the start. Again, it should be noted that although the full BTeV event reconstruction code is developed and coded as part of the L3 Trigger software project, the L3 trigger does not have to run every single component that is normally desired in a real full offline package.

It is reasonable to assume that attention to code speed can be achieved since the traditionally offline reconstruction code for BTeV is all developed as part of the L3 Trigger software project, rather than a much more open-ended offline project. At the beginning of data taking the emphasis will not necessarily be on the fastest processing speed for the full event reconstruction code. At lower luminosity, as long as the efficiency and rejection goals

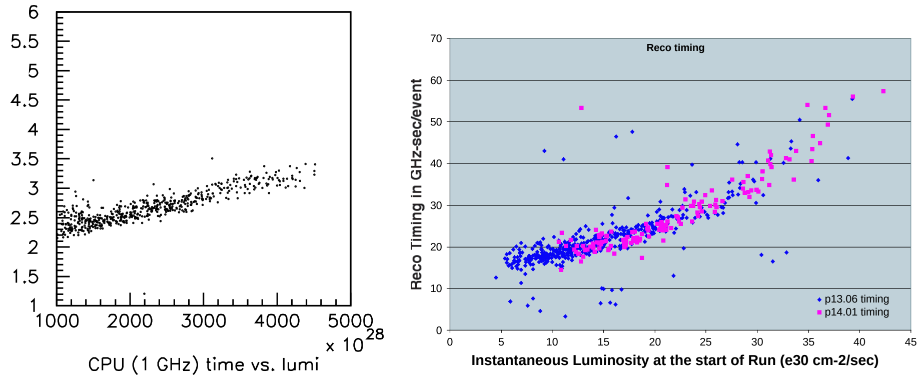

Figure 11.30: Offline processing time per crossing for (left) CDF and (right) D0, plotted as a function of luminosity.

of the L3 Trigger are met the full event reconstruction processing could take longer than 134 ms/event. At this point the code would still be developed and tweaked for optimization of the physics goals. We would run as much of the full event reconstruction code as needed to achieve the L3 Trigger requirements. There is considerable scope contingency in the L3 trigger since not everything included in a full "offline-like" production need be run at L3. As the luminosity increases we can be certain to reduce the latency for each piece of the full event reconstruction code, and thus be able to run more components of this in the L3 Trigger while keeping the latency to less than 134 ms/event/CPU. The rejection requirement for L3 is only a factor of 2 reduction compared to the L2 rate. In addition there is flexibility in optimizing the rejection at each stage of the trigger, so that if necessary either L1 or L2 could provide extra rejection.

Although the full "offline" reconstruction code is developed and written as part of the L3 Trigger software project, we are not planning on necessarily running all of it at the L3 Trigger stage. Nevertheless we have shown that it is possible, even likely, that the majority of the full event reconstruction can be run at the L3 Trigger stage. At the beginning of the experiment when reconstruction code and calibration procedures are being ironed out, some offline processing or reprocessing will need to be done. Some of this processing could be done on free cycles of the L2/3 PC farm, since we assume a duty cycle of 33% for beam (when averaged over long periods), or it could be done at PC farms at collaborating universities. As the code matures and is optimized for speed, more and more of the "offline" reconstruction would actually be run at the L3 Trigger stage. Some offline processing would probably always be desirable, like splitting off different physics data streams, or adding additional interesting physics streams. The data-taking duty cycle is low enough that the L2/3 PC farm should be available a fair fraction of the time for this sort of offline processing. In addition university groups have large PC processing farms that could be used.

Data Event Size and Data Storage

One of the challenges in the BTeV Trigger system is to reduce the total output data rate from L3 to 200MB/s. At a crossing interval of 396 ns, this corresponds to reducing the event size from an estimated 250 KB to 80 KB in the L3 Trigger. This is achievable because the objective is to run the equivalent of components of the full offline event reconstruction at L3 and to replace some of the raw data with DST ("Data Summary Tape") data as output. The DST output data for a detector subsystem contains all the information needed for physics analyses but does not contain enough information for full reprocessing (re-reconstruction) of that subsystem. The event size reduction will be achieved in several stages over time as the luminosity increases. The real requirement on the L3 Trigger is an output rate of 200 MB/s or less. At full luminosity, this corresponds to an event rate of 2.5 KHz and an event size of 80 KB. A small percentage of (prescaled) triggers of larger event size with more event information will also be written out for monitoring and trigger studies. We expect that as the experiment (detectors and software) matures we will be able to write more of each subsystem information in DST data format at the L3 Trigger.

		Event Size (KBytes)							
	#Bytes/	**2 ints/BCO**		**4 ints/BCO**		**6 ints/BCO**		**9 ints/BCO**	
	hit	**L1 in**	**L1 out**	**L1 in**	**L1 out**	**L1 in**	**L1 out**	**L1 in**	**L1 out**
Pixels	7	13.3	21.4	25.1	38.8	36.7	46.1	53.2	55.3
Straws	4	4.2	7.1	8.0	11.7	11.5	15.2	16.7	18.2
FSil	3	0.6	1.0	1.2	1.5	1.6	2.2	2.3	2.6
RICH	3	4.2	7.3	8.4	12.3	12.3	16.0	17.8	22.8
ECal	4	4.2	5.5	6.0	8.1	8.1	9.8	11.8	11.8
Muon	2	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.5
Total		26	43	49	73	71	90	103	108

Table 11.10: Estimated event (BCO) sizes from projected numbers of bits for each detector and using the occupancies given by the BTeV Geant simulation for different average numbers of interactions per bunch crossing.

The raw event size of 200 KB is a somewhat conservative estimate based on simulations of the BTeV spectrometer running at full luminosity. The occupancies and the projected numbers of bits per hit for each detector subsystem were used as input, the results are tabulated in Table 11.10 for running at different average numbers of interactions per bunch crossing. With an average of 9 interactions per crossing, the average size of a crossing (event) input into the L1 Trigger is determined to be $\approx$ 100 KB, and the output from L1 is $\approx$ 108 KB/event. We have taken twice these numbers to get 200 KB and 250 KB for the event size input and output of L1. This is thus a conservative estimate and it allows for possible extra noise hits and beam related backgrounds, and provides some extra headroom (over capacity). For comparison, the average raw Run 2 D0 event size is about 170 KB, while the CDF event size input into their offline farm is about 220 KB.

To illustrate the size of the likely data reduction we have considered the possible event sizes when part or all of the charged particle tracking is done at L3. This was done since the results for preliminary L3 tracking (see Table 11.7) show it is possible to do charged particle tracking at L3. With full charged particle tracking done one could eliminate the raw data from the pixel, straw and forward silicon detectors. Even with just pixel-only tracking done the event size would be significantly smaller by eliminating the raw pixel data. The estimated event sizes (including headroom as explained above) are given in Table 11.11. Note that the options listed in Table 11.11 are just some of those that could be used for data reduction. Other options can include for example keeping some of the raw pixel or tracking hits that are associated with particular tracks, or keeping some hit clusters but dropping raw hits. The options will become clearer as the L3 software projects evolve.

Description	**Event Size (KBytes)**
Into L1	200
Out of L1	250
Out of L1 + L1, GL1 info	255
At L2 event assembled	201
Out of L2 + L2 pixel tracks, vertices info	208
Out of L3 + charged track, neutral Vee L3 info	211
Out of L3 without raw pixel data	140
Out of L3 without raw pixel, straw, FSil data	100
Out of L3 no raw pixel/straw/FSil data + compressed	75

Table 11.11: Estimated event (BCO) sizes used for the design of the BTeV Trigger. The event sizes included some headroom as explained in the text and are given for different stages of data reduction. Note that these are just some options for data reduction that would be available, others are discussed in the later section on risk mitigation.

The average L3 Trigger output event size of 80 KB is thus thought to be achievable over the initial commissioning of the experiment. It will be most important when the full luminosity is reached. As the experiment matures we will be able to drop more and more of the raw data and write out instead high-level information (DSTs.) These DSTs will contain all the information necessary to do any physics analysis but will not contain enough information to fully reprocess or re-reconstruct the events. Although this might appear somewhat risky, all raw data will be kept at the beginning of the experiment. Later, as the Level 3 Trigger code and calibration procedures mature, less and less of this raw data will be needed for physics analyses. This is actually thought to be necessary since just the large amount of data suggests that reprocessing is unlikely to be an easy option. In fact, in our experience, past fixed-target experiments that produced vast amounts of data did not reprocess their full data sets.

Finally as a comparison, the FOCUS experiment has DSTs with 3 KB/event. If we normalize to the primary multiplicity (17/4) and to the average multiplicities for B compared to charm decays (5/1.8), we would project a DST event size in BTeV of about 35 KB. Instead if we take a perhaps better comparison, the D0 experiment has DSTs that are about 125 KB/event. However their DSTs actually contain enough information for partial reprocessing, (which includes rerunning the calorimeter clustering and track finding and fitting.)

Without this lower level information their DST size could be much smaller. For example the D0 “Thumbnail” data set which only contains very high level information that is efficiently packed is 5 KB/event. Similarly CDF produces a DST from their offline reconstruction that contains both raw and reconstructed data with an event size of about 300 KB. CDF is switching to a mode where they will write a DST with an event size of 170–190 KB. For physics analyses, CDF produces an “ntuple-type micro-DST” format that contains some raw data and averages about 30 KB/event.

Besides reducing the amount of information, some reduction can also be achieved with compression if necessary. For example, the GNU “gzip” utility can reduce the size of the FOCUS DSTs by more than 50%. The trade-off is that this would require additional CPU processing time. The event sizes quoted for D0 already include significant compression as part of their normal processing. The 220 KB CDF event size out of their L3 trigger farm can be compressed to about 160 KB.

Each L2/3 Trigger Farm worker PC and Manager PC will include large disks as part of a normal purchase. These will act as data caches to ensure good network data transfer efficiency and achieve reliability and fault tolerance of the system. For example, a 400 GB disk on each PC could buffer data input to the L2 trigger for 6.8 hours. Alternatively, if needed as an output buffer, a 400 GB disk on each PC could buffer the output from L3 for over 2 weeks. The disks on the Manager PC’s will be used as database caches, output event pool caches for monitoring or trigger studies, and for use in offline processing.

11.6.1.2 Prepilot, Pilot and Production Hardware

A prepilot L2/3 trigger PC farm is being built using 32 dual processor worker PC’s and a dual processor Manager PC in FY04. The worker PC’s are a mixture of old Fermilab Computing Division Farm PC’s. These consist of a regular ATX desktop case each containing two 333MHz Pentium II CPUs, or dual 500MHz Pentium III CPUs. (These are sufficient for development purposes, but may be expanded in FY05 using additional retired Fermilab Farm PC’s if available.) The worker nodes are networked via two 10/100 switches to a Manager PC and a Data Server simulating the function of the L1 buffer. The Manager PC and Data Server are connected to the switches via 1000Base-T gigabit ethernet ports. This prepilot PC farm is being used for a number of development projects for the L2/3 Trigger including the RTES 2004 demo project. It will be used to try out different infrastructure software to handle and manage farm worker processing, such as FBSNG - the farm batch tools (http://www-isd.fnal.gov/fbsng) used by the Computing Division Farms group. It will be used for developing infrastructure L2/3 Trigger and DAQ specific code, and testing the monitoring and control code from the RTES project.

The prepilot L2/3 Farm will also be used as one of the test beds for developing and testing software that will be needed to use the L2/3 PC farm as an offline processing resource, for example Grid-related software to enable part of the farm for Monte Carlo simulations or data processing when processor cycles are free. The prepilot L2/3 farm will also be used to develop more reliable and manageable worker nodes, e.g. by investigating the effect of diskless operation or using solid state disks for the OS and system related software, or other minor hardware modifications for monitoring purposes. The software development work will be carried out by the BTeV Trigger team and members of the RTES group, together with some consultation with the Fermilab Computing Division, including the Farms group, the CDF/D0 Reconstruction groups and the QCD Lattice engineering group. A more powerful pilot L2/3 PC farm will be built consisting of about 5% of the baseline farm (in complexity, not compute power). These farms will be used for continuing software development and coding. If blade servers look to be a viable alternative, the pilot PC farm may consist of this technology for evaluation and development.

Since the L2/3 Trigger farm and DAQ infrastructure will not be purchased until relatively late in the project, simulations will have to be used for queuing studies. These studies will be carried out in the first two years as part of the development work needed to decide on the L2/3 Trigger farm design and technologies. Even without a large PC farm, individual hardware components can be tested to ensure that specifications can be met. In addition the prepilot farm can be used for system queuing studies. When the pilot farm is available, these queuing and network throughput studies will continue on more up-to-date hardware, leading up to several data challenges closer to the end of the project.

11.6.2 L2/3 Software

The L2/3 trigger is a farm of commodity processors running a POSIX compliant, open software operating system. In the baseline the PC farm runs Fermi Linux, produced by the Fermilab Computing Division. After an L1 accept all data for the event will be transferred from the Level-1 buffers to a L2/3 processor. If the event passes the L2 trigger it is then processed by L3 in the same farm node.

The distinction between L2 and L3 is that L2 uses only a subset of the data whereas L3 uses the full event data. In our baseline, the L2 trigger is solely based on the pixel data and consists of a sophisticated tracking and vertexing package, designed to reject most of the bunch crossings without heavy quark decays. However, since the data acquisition architecture delivers the entire event for all L1 accepts, we will have the possibility of creating other L2 triggers, such as an L2 Muon trigger.

The basic requirements for the L2 trigger are (i) a rejection factor of 10 on light quark crossings (ii) an acceptance higher than 90% on relevant heavy quark decays (iii) takes 5 msec or less per bunch crossing on a “12 GHz P4 CPU”. We do not anticipate that memory utilization will be a critical issue.

The input data to the L2 vertex trigger consists of all L1 tracks and vertices and the raw pixel hits. The L2 algorithm performs Kalman filter track fits on the L1 tracks and refits the primary vertices. The kalman filter code is discussed in more detail in Section 11.6.2.5. It then searches for other primary vertices and detached secondary vertices. It also looks for high p_T single detached tracks, corresponding to decay modes with only one charged prong (e.g. $B^+ \to \pi^+ \pi^0$).

The goal of L3 is to achieve another factor of 2 in background rejection and to reduce the size of the event by a factor of 3. The L3 algorithm will use components of the full offline reconstruction code that is all developed and written as part of the L3 software project. We have prototype code for forward tracking, K_s reconstruction, particle ID in the RICH, and electron, photon and π^0 reconstruction in the calorimeter. The CPU and memory requirement for this L3 stage is specified in the requirements so that software developers can pay proper attention to performance issues. The L3 code must not consume more than 134 ms per event per CPU to perform all reconstruction and event formatting in a PC with 1 GB of RAM.

11.6.2.1 The L2/L3 Software Architecture

The L2 and L3 algorithms described below rely on good calibration and alignment data. Some of this data is in fact a by-product of the reconstruction code running inside these L2 and L3 filter programs. One must also monitor the performance of these reconstruction codes and control the trigger tables. This adds to the complexity. The system architecture is shown in Fig. 11.31. This block diagram presents the overall architecture with respect to the DAQ and global trigger. Details are presented in Fig. 11.32 and 11.33. Also shown in Fig. 11.32 is a view of activities that take place inside a typical worker node. The last figure shows the tasks undertaken by the L2/3 filter program(s). This is a conceptual view of the software: boxes do not map one to one to a given set of Unix processes. This mapping is outside the scope of this document.

Most of the elements shown in Fig. 11.31 have been already discussed above. The trigger components are connected to the outside world via the DAQ Elements for controls and via the database interfaces to get calibration and alignment data, specific sub-detector configurations and high level summaries of performance measurements. The L2/3 worker nodes will also generate new calibration and alignment data which will have to be uploaded in the calibration database. This information will in turn be disseminated to other worker nodes in the system. The main output from the worker nodes consist of streams of reconstructed data, to be collected by the outside world (see Fig. 11.32).

The manager nodes are different than the more numerous workers nodes. Their function is mainly administrative: they will keep and disseminate the configuration information, state notifications and information about fault or error conditions and log files book-keeping. In addition, they will collect and summarize the calibration and alignment data from the worker node and periodically re-distributed it to them. For a given partition, there is a 3-level hierarchical organization among nodes: BTeV operators will download instructions to the Global manager, which in turn distributes them to the manager nodes, and finally the managers pass these directives to the worker nodes.

The L2/3 filtering programs shown in Fig. 11.33 are also complex entities. The physics algorithm plays the key role. They must be supported by various services to function properly, such as the fixed, nominal geometry settings. Since they consume and generate alignment and calibration data, data path to databases and management PCs will be provided. As we plan to have the possibility of running multiple L2 or L3 physics trigger algorithms, a control unit must be provided to manage the multiple “Trigger Paths”, so that the integrity of the information is preserved and optimized to avoid redundant calculation. Finally, an important by-product of these algorithms is accelerator, or beam physics data, such as average beam positions, bunch lengths and crossing angles. Conversely, as part of the overall monitoring task, the L2/3 algorithms use accelerator control data, such as bunch intensities to check relative bunch crossing rates. A dedicated data path[3] to deliver this data to our colleagues from the Accelerator Division will be part of the architecture.

11.6.2.2 L2 Algorithm

The L2 pixel algorithm has basically two distinct components:

- Refinement of the L1 pixel tracks using a Kalman fit, pointing to the relevant L1 vertex that triggered the event.

- Search for primary vertices, secondary vertices and isolated detached tracks based on L2 tracks.

Given the basic feature of the pixel detector, we perform tracking/vertexing pattern recognition and fitting in three dimensions at all stages of the event reconstruction. Tracks are assumed to be straight lines in the non-bend plane (horizontal) and near perfect circles in the vertical plane. Unlike L1, track and vertex error matrices are estimated rigorously based on the known pixel resolution and the track momenta, that is, multiple scattering is always taken into account in the correlated position error calculation.

Currently only L1 tracks and the associated hits from the inner and outer triplets are used in the L2 fits. If it is found necessary to improve the L2 efficiency, extra hits on these tracks can be searched for. In addition extra tracks can be reconstructed from the unused hits.

The Kalman filter is the track fitting engine that returns a confidence level based on hit position errors and the best track parameters either at the vertex, or at any arbitrary extrapolated (or interpolated) position. This fitting method is fairly rigorous but quite CPU intensive. The implementation can be fast because we can assume uniform magnetic field, or, if corrections are needed for long and/or low momentum tracks, such corrections are small and can be treated as perturbations.

The trigger strategy is based on the fact that the primary vertex has at least 5 tracks. L2 tracks are always associated with at least one primary vertex. The primary vertex that the L1 trigger associated with detached tracks is used as a seed for the L2 primary vertex.

Vertices are constructed based on a linear approximation of the trajectories near the beam line. That is, we use the Kalman filter to extrapolate the track close to the beam line, then perform the vertex reconstruction using these track parameters entirely ignoring

[3] Currently, the Accelerator Control system uses ACNET. These data are accessible via ethernet through existing TCP/IP interface routines.

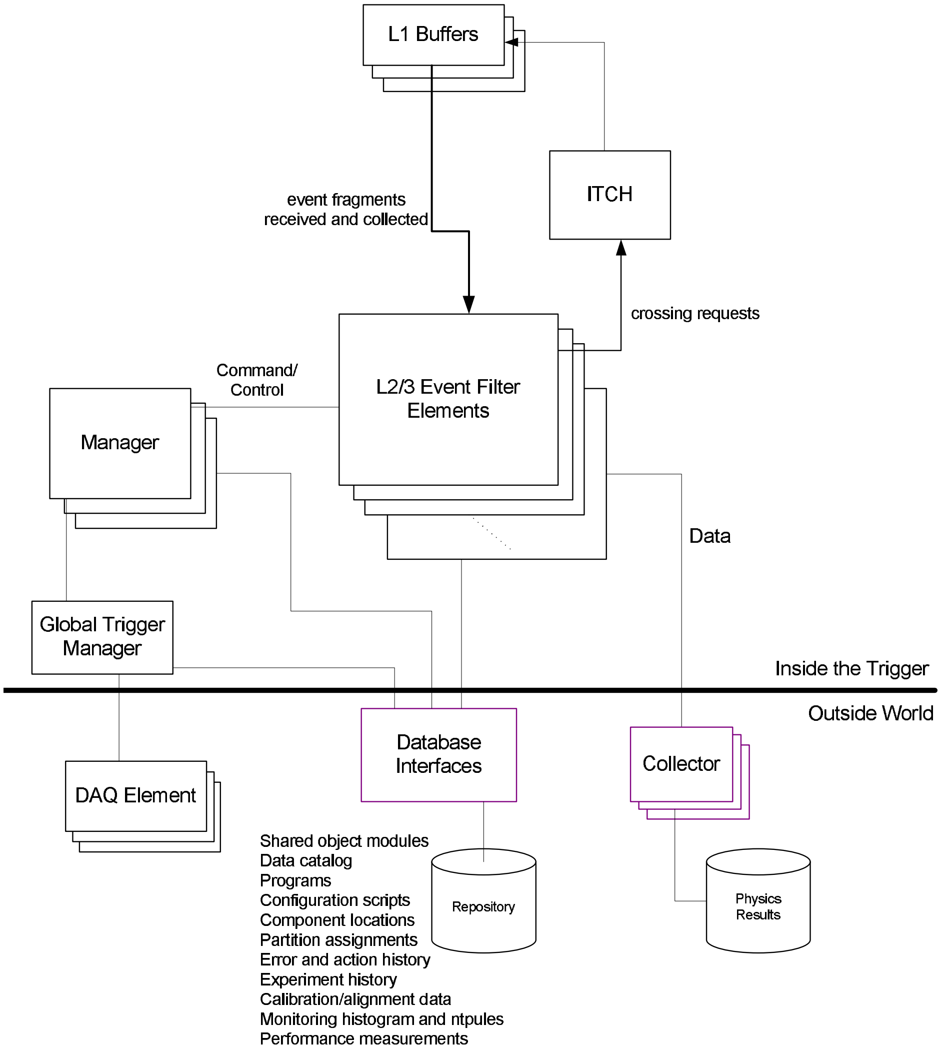

Figure 11.31: The L2/3 software architecture in relation to the data acquisition and overall trigger.

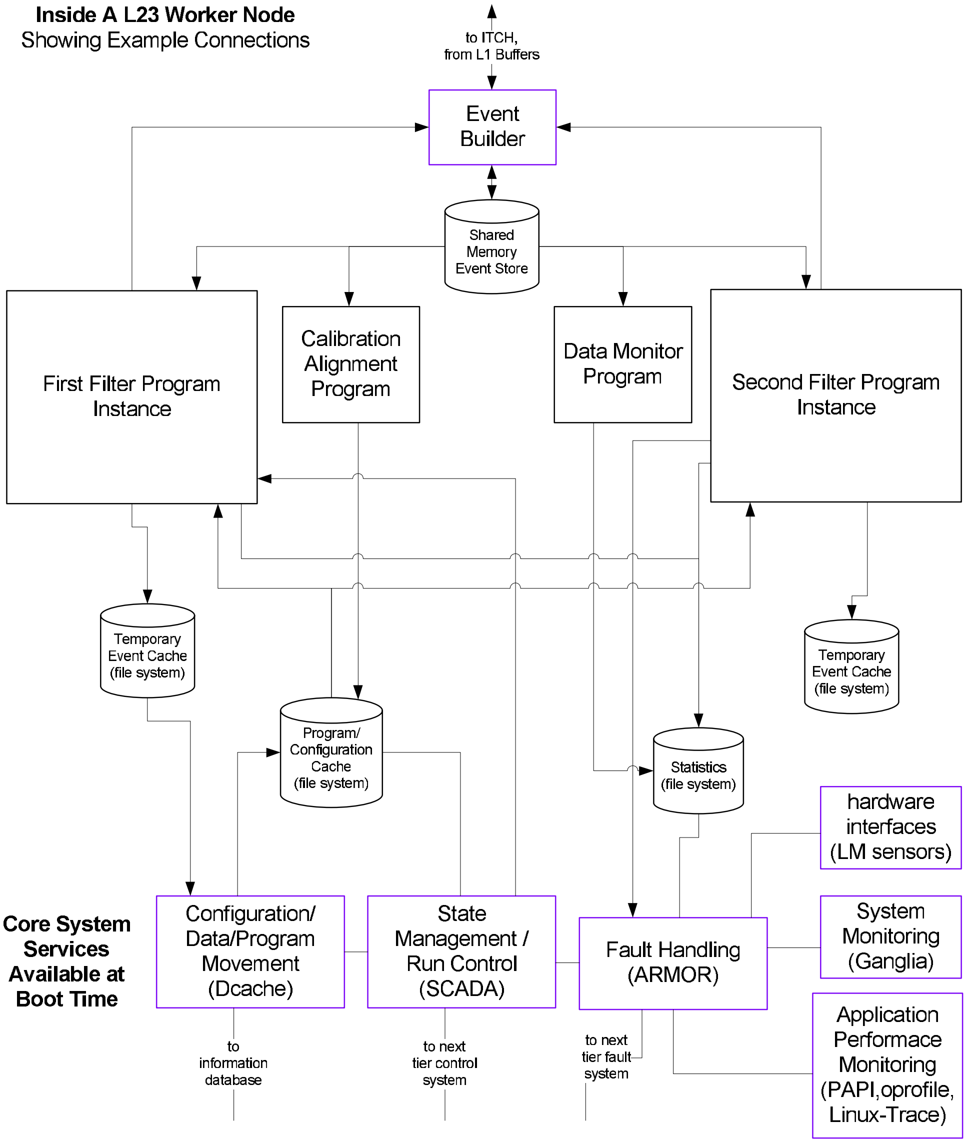
Inside A L23 Worker Node
Showing Example Connections
to ITCH,
from L1 Buffers
Event Builder
Shared Memory Event Store
First Filter Program Instance
Calibration Alignment Program
Data Monitor Program
Second Filter Program Instance
Temporary Event Cache (file system)
Program/ Configuration Cache (file system)
Temporary Event Cache (file system)
Statistics (file system)
hardware interfaces (LM sensors)
Core System Services Available at Boot Time
Configuration/ Data/Program Movement (Dcache)
State Management / Run Control (SCADA)
Fault Handling (ARMOR)
System Monitoring (Ganglia)
Application Performace Monitoring (PAPI,oprofile, Linux-Trace)
to information database
to next tier control system
to next tier fault system

Figure 11.32: Conceptual view of tasks undertaken by the L2/3 worker nodes.

the magnetic field. If need be, the tracks can be extrapolated again to the fitted Z vertex position. This iterative procedure converges very fast, and we do not need to compute intersection of circles, which is a bit more complicated than lines.

The trigger decision is based on transverse momentum and detachment criteria. A secondary vertex must satisfy the following criteria: (i) tracks must have a confidence level greater than 2.5%, must be detached from the primary vertex by more than 3.5 σ and must have transverse momentum > 0.5 GeV; (ii) all such detached tracks must have the same sign p_z and be pointing away from the primary vertex (to reject nearby primary vertices); (iii) the

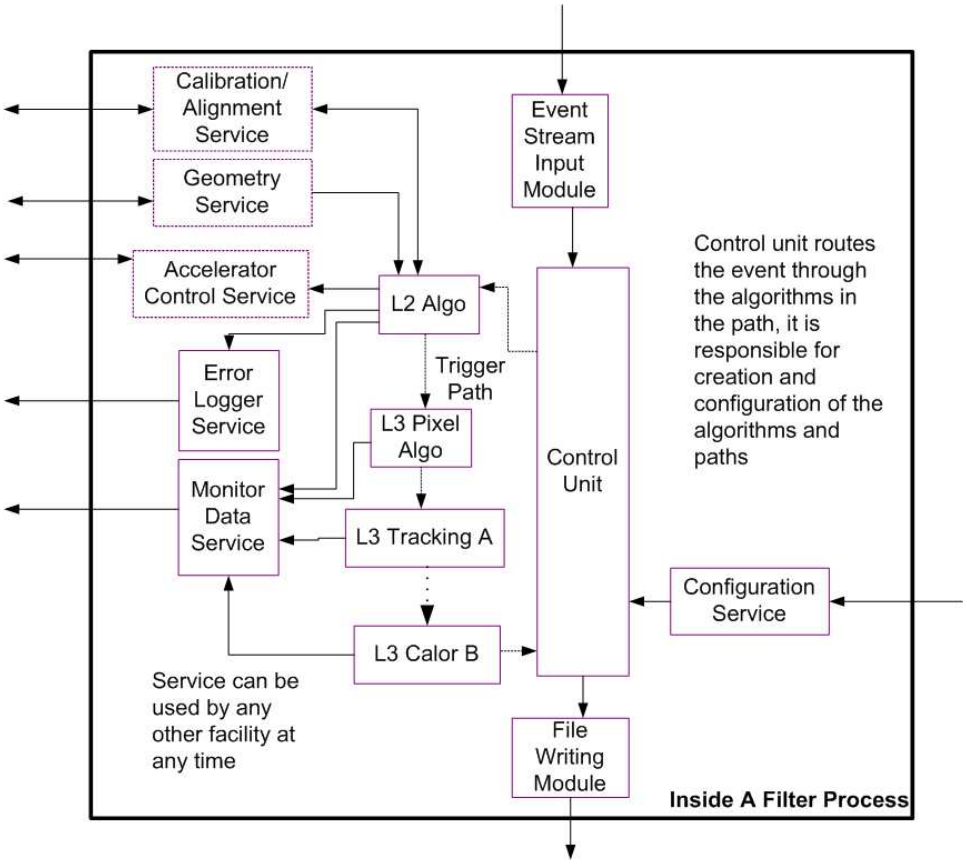
Calibration/ Alignment Service
Geometry Service
Accelerator Control Service
L2 Algo
Error Logger Service
Monitor Data Service
Trigger Path
L3 Pixel Algo
L3 Tracking A
L3 Calor B
Event Stream Input Module
Control Unit
Control unit routes the event through the algorithms in the path, it is responsible for creation and configuration of the algorithms and paths
Configuration Service
File Writing Module
Service can be used by any other facility at any time
Inside A Filter Process

Figure 11.33: Block diagram of the tasks occurring inside the L2/3 filter programs.

secondary vertex must have a confidence level greater than 2% and must be detached from the primary vertex by more than 3.5 σ ; (iv) the secondary vertex must have an invariant mass less than 7 GeV and more than 100 MeV outside the K_s mass. An event passes L2 if it has either a detached secondary vertex or a high p_T detached track.

As the pixel planes need to be retracted while we load the Tevatron, we need to re-align the pixel detector at the beginning of each store. The pixel detector alignment will be based both on accurate position sensor information and on tracks coming from minimum bias interactions emerging from the luminous region. Again, the Kalman filter will be the workhorse of the alignment package. Pattern recognition and Kalman fits will be performed using hits with errors based on estimated alignment errors. The Kalman filter is able to report deviations of the hits with respect to the best approximation of the trajectory, leading us to an iterative algorithm. We will assume that each pixel half-plane moves as a unit; we do not plan to find alignment constants for each individual pixel at the beginning of each store.

Event type	L2/L1
Light quark	7%
$B_s \rightarrow D_s K$	85%
$B^0 \rightarrow \pi^+\pi^-$	87%
$B^0 \rightarrow J/\psi K_s$	78%
$B^- \rightarrow \pi^- K_s$	72%

Table 11.12: L2 trigger efficiencies

11.6.2.3 L2 Implementation

The L2 code will be written in C++. The production code will then be optimized for speed, and will be tightly interfaced to the data acquisition software. There will be no unnecessary re-copying of the raw pixel data to the L2 internal buffer, since judicious use of pointers will be used instead ("shallow" copies).

The L2 vertex trigger code will be tightly coupled to the pixel alignment code. It will share the same interfaces to raw data and to the Kalman filter package. In addition, the L2 code initialization will be able to access the BTeV Geometry Database upon demand. The Pixel Alignment code, running concurrently with the L2 code on the L2/3 farm, will generate new alignment data periodically (for example each time the pixel detector moves, after Tevatron injection).

We will maintain and upgrade the interface between the BTeV Monte Carlo code and the L2 trigger. That is, all modifications of the L2 code will be first tested on Monte Carlo data. A Monte Carlo representing the "as built" version of the detector is therefore mandatory. The L2 code will be able to assemble and fit a given track using the exact hit list generated by the Monte Carlo. Finally, the L2 code will also be callable from the off-line code, and can be re-run on selected raw data.

11.6.2.4 L2 Simulations

The current L2 vertex trigger code has been run on 4 different decay modes representing different decay topologies, and on light quark events. The timing results are shown in Section 11.6.1. The current code runs over 7 times faster than our requirements so we have plenty of leeway to improve the efficiency by refining the tracking and vertexing algorithms or adding extra algorithms such as a L2 muon trigger. The efficiency results are shown in Table 11.12.

11.6.2.5 L3 Algorithms

The goal of the L3 software project is to fully reconstruct the heavy quark decays and characterize them. This does not necessarily mean reconstructing the entire event at L3: for instance, if the vertex pattern and reconstructed mass is consistent with an all charged exclusive decay, the time-consuming electromagnetic shower reconstruction phase can be skipped. The L3 event reconstruction stage will be based on results from L2, raw hit data and the alignment/calibration constants from the database. The output will consist of a semi-inclusive list of 3- or 4-momentum vectors (*i.e.* with or without particle ID), and vertex patterns. Additional information will also be made available for further off-line analysis, such as a partial list of hits or ADC values associated to critical tracks. We are not planning to transfer the entire raw data and reconstructed data to permanent storage media. However, during the commissioning period, there will be enough disk available on the L2/3 farm to store ≈ weeks of raw data, for subsequent re-analysis.

During the pre-conceptual R&D phase of BTeV, we wrote prototype reconstruction code for all sub-detectors (except the Muon detector). We now describe the essential features of these reconstruction codes, or refer to the detector specific section.

Forward Track Reconstruction

The Forward Silicon and the Straw detector are analyzed jointly. The goals of this reconstruction phase are to:

- improve the momentum determination
- improve the vertical track position at the vertex
- provide accurate track position measurements at critical locations for particle identification (at the entrance and mirror positions of the RICH, at the E.M. calorimeter and at the muon wall.
- reconstruct K_s and Λs

An L3 charged track can emerge from the pixel detector, or from the forward tracking system. Tracks emerging from the forward tracking system may be tracks from the $p\bar{p}$ collision that do not have enough hits in the pixel detector to form a reconstructible pixel track, or from decays of K_s and Λ^0, or from reinteractions. Tracks from reinteractions could be of interest because they can potentially confuse the pattern recognition within the forward tracking detector itself, or in the RICH, muon detector or the calorimeter. They will be reconstructed last, on an as needed basis.

The Forward Silicon and Straw detectors are handled jointly. There are no distinct "Silicon tracks" and "Straw tracks". This is because most of the tracks encounter first a few silicon detectors, graze the hole in a straw station and finally emerge as "Straw tracks". Since there is an overlap between the Silicon and the Straw detectors, a track at any given station can have hits in both the Forward Silicon and Straw detectors.

The last tracking station, located between the electromagnetic calorimeter and the RICH detector, plays a different role from the others: its sole purpose is to pin-point the track at the RICH mirror and/or at the front of the calorimeter. Multiple scattering is such that this additional measurement point does not improve the knowledge of track direction upstream of the RICH.

Unlike the pixel detector, the forward tracking pattern recognition must recognize cases where multiple tracks go through the same cell, because the solid angle covered by such cells is much larger. Thus, it is a much more involved pattern recognition problem. Most tracks of real interest are already fairly well defined as they emerge from the pixel region. For such tracks, it is simply a matter of allocating the correct set of Silicon or Straw hits along their path, refitting to improve the track parameters. Once that part is done, only about half of the hits remain unused. These hits are used to reconstruct K_s, Λ ("Vees") and background tracks.

A prototype for this first stage has been written. The reconstruction of the forward tracks coming from the interaction region is seeded by the L2 pixel tracks. Tracks are propagated to the next encountered station (Silicon or Straw), a list of candidates hits for each plane is built, and Kalman fits are performed for each potential combination. For the straws, we fit a set of 2 or 3 hits within one stereo view to a given track rather than individual hits to avoid unnecessary combinatorics of many possible Kalman fits. Thus, left-right ambiguities are usually lifted prior to fitting. Arbitration can be postponed until we reach the next station downstream, if not enough hits are found in a given station, for instance. Although rather CPU intensive, this procedure converges in the sense that only one set of hits remains when the track reaches the RICH (or leaves the spectrometer). In many cases, the arbitration must be postponed until the next stations are reached, because a Straw hit can be "overwritten" by another track, biasing the measured track position.

The preliminary tracking efficiency versus momentum, for tracks reaching the RICH, is shown in Fig. 11.34. About half of the inefficiency is due to inaccuracies in the multiple-scattering accounting in the Kalman fits and the other half is due to pattern recognition confusion or double occupancy in the straws. The momentum resolution obtained via this full pattern recognition is in very good agreement with the fits performed in the context of BTeVGeant, where all hits are always assigned to their respective tracks. Once station 6 (located in front of the RICH) is reached, the probability of accepting "ghost" tracks is quite small, about 0.5%.

The reconstruction efficiency for low momentum tracks curling in the downstream end of the magnet and leaving the spectrometer between the first two stations will definitely be lower than for tracks reaching the downstream end of the spectrometer. By reconstruction efficiency, we mean here the probability to obtain a set of track parameters consistent with the real one, not merely the probability of finding a matching set of hits that seem to fit the track.

Unlike in the pixel detector, the magnetic field can no longer be assumed uniform and of constant direction. Exact numerical integration of the field along Z in the Kalman fit extrapolation routine requires too much CPU time. Therefore there will be a set of fast parametric extrapolation routines based on tabular data describing integrals of fields (or "moments").

Kalman Filter

The BTeV software suite includes prototype Kalman filter code which is currently used

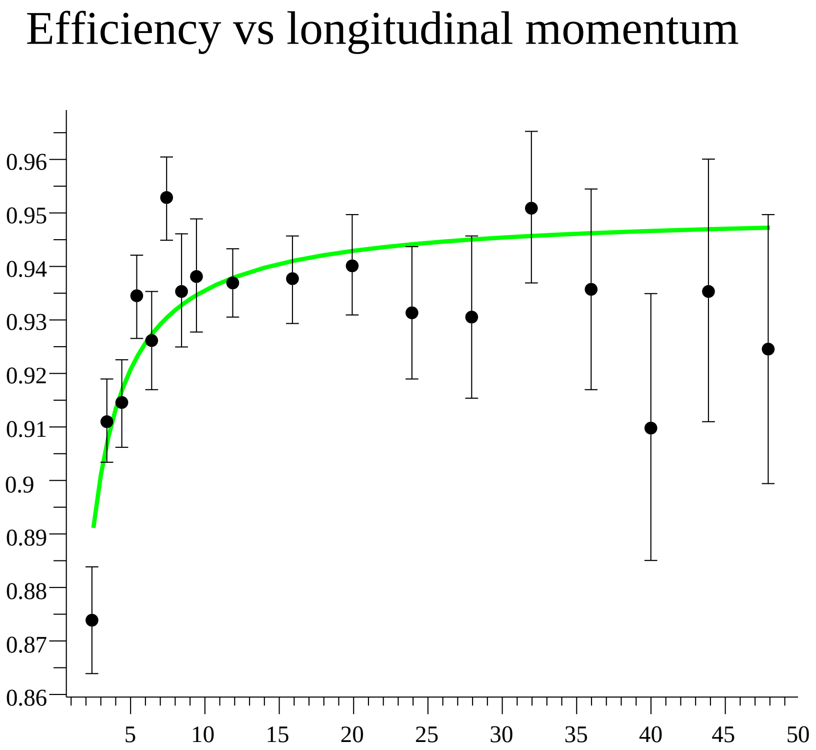

Figure 11.34: The preliminary efficiency versus the longitudinal momentum P_z (GeV/c) for the L2-seeded (pixel-seeded) L3 tracks. The fit is simply there to guide the eyes, the function is $E_\infty * (1 - k/P_z)$. The parameter k is statistically significant, indicating problems at low momentum. This is where our geometrical acceptance drops sharply. Note that the loss of "efficiency" includes particles lost through interaction in the material of the detector and due to (recoverable) problems with the current detailed description of multiple scattering sources in the analysis program.

as the final track fitter for the simulated physics analyses of BTeVGeant output, for the L2 prototype code and for the forward track reconstruction package. It is also used by the RICH code to interpolate/extrapolate tracks into the RICH detector and by the electromagnetic calorimeter code to extrapolate tracks to the face of the electromagnetic calorimeter for track-shower matching. At present the Kalman filter code works only as a final fitter - codes which use it must supply it with a list of hits which are to be considered as a single track. These hit lists are provided by different codes for the different uses of the Kalman filter and those codes are described in the corresponding sections of this report, for example the previous section which described the forward tracking code.

The Kalman filter code automatically includes multiple scattering and energy loss for the material associated with each hit in its hit list. That is, it knows the amount of multiple scattering associated with the material in a single pixel station, a single forward silicon plane or a forward straw plane. Auxiliary routines, not part of the Kalman filter package, discover additional material through which the trajectory of the track passes, such as detectors with missing hits, and inert material, such as beam pipes, and pressure vessel walls. These auxiliary routines need to be improved to understand more of the details of the support structures in the detector.

The Kalman filter code can be asked to filter in several different directions. It can run from the start of the track to the end or from the end back to the start. It can also fit in both directions to produce smoothed estimators of the trajectory at any point along the trajectory. This feature is used by the alignment code to compute residuals at each hit.

The Kalman filter code also has interfaces for adding new hits at the start or end of an existing fitted track. At present this interface is primitive and difficult to use. There is work planned to improve the interface so that it is both easy and natural to use the Kalman filter within pattern recognition codes. When that work is done many optimizations will be possible in the L2 code and in the forward tracking code.

The Kalman filter code can select from several algorithms for propagation of the track and its covariance matrix through magnetic fields. This allows fast propagation code to be used when speed is required, such as in L2, and slower, but more precise, propagation code to be used for the final fits before tracks are used in physics analyses. At present only two choices of field integrators are available, representing the extremes of fast but approximate and precise but slow. It is envisaged that several intermediate solutions will be developed.

Several other optimizations are under consideration to speed up the operation of the Kalman filter when it is used in the L2 trigger. For example, some of the matrix multiplications involve matrices with small off-diagonal elements. In certain cases these can be dropped and the matrix operations hand-coded to exploit the resulting zeros.

Preliminary K_s^0 Tracking Studies

We now describe the reconstruction of the K_s^0 for which we have no (or not enough) pixel hits. From a detailed study of the track topology of these $\pi^+ \pi^-$ pairs, we conclude that the largest reconstructible sample consists of tracks reaching straw station 6, for which we have 3 consecutive straw stations in the nearly field-free region beginning at $z \approx 2.75$ meters from the magnet center. Stations 4, 5 and 6 are located at $z \approx 2.9$, 3.3 and 3.8 meters from the magnet center, respectively. The following algorithm has been partly coded and is currently under study:

- Selection of “un-used” straw hits. We mark all the hits used in the above pixel-seeded L3 tracks, as “used”, thereby getting rid of about 1/3 to 1/2 of the available hits.

- Reconstruction of straw hit triplets (or doublets) within a straw stack (or “view”). Despite the lack of good constraints from unknown track slopes, the multiplicity of such small, 2D tracks within a station and a stack is not overwhelmingly large.

- Reconstruction of 2D tracks between stations 4, 5 and 6, for each stereo view.

- 3D track matching of the 2D views.

- First reconstruction of a 3D K_s^0 vertex using the non-bend plane 2D vertex as a seed. The K_s^0 trajectory is constrained: it must come from the selected L2 primary vertex for which we have good detached p_t. This allows us to obtain a preliminary determination of the $\pi^+ \pi^-$ and K_s^0 momenta as well as the K_s^0 mass.
- Search for confirming hits in upstream stations, followed by track and vertex refits.

Preliminary studies indicate that we can reconstruct K_s^0's that decay upstream of station 3 with about 60% efficiency. The loss of signal is mostly due to the high occupancy in the straws.

RICH Particle Identification

The RICH reconstruction code will be based on the algorithms developed to assess the performance of the detector. We note here that all charged tracks entering the RICH volume must be reconstructed so that all Cherenkov rings can be characterized based on the pion hypothesis. This is the starting point in the algorithm, as there are too many interfering hits in the 3σ ring to identify a given track independently from all the others. The output of the RICH code is a set of electron, muon, pion, kaon and proton identification probabilities for each track.

Numerous high momentum charged particles, particularly electrons, that are created in the detector material (pixel, forward silicon, beam pipe, etc.), can produce light in the RICH. Hence, we are writing a dedicated pattern recognition code aimed at reconstructing rings emitted by such high velocity particles, in order to mitigate this occupancy problem. Hits and tracks segments from Straw and Forward Silicon stations 5, 6 and 7 could also be helpful at determining such background rings.

Muon Reconstruction

The Muon reconstruction will be based on the algorithm developed for the L1 trigger, and on the knowledge of the parameters of tracks emerging from Station 7. Thus the muon code will mostly consist of track segment matching.

E.M. Reconstruction

As part of the design effort and detailed evaluation of the expected performance of the detector, a stand-alone reconstruction code has been written. As for the previous code, it is assumed that the charged particles hitting the detector are known and characterized.

Heavy Quark Filter

We require the L3 trigger to reject about 50% of the events that pass L2 in order to achieve an output rate of 2.5 KHz when running with an average of 6 interactions per bunch crossing. In order to be as efficient as possible for known decay modes of interest yet keep the trigger as open as possible to new ideas we adopt the following strategy. We conduct a search through a list of decay modes of interest, using as much information from the sub-detectors as necessary to test the relevant hypothesis. These events are selected with highest priority. We then select events with evidence of a heavy quark vertex. We expect the rate of events with reconstructible b-quark decays to reach about 1 KHz when running at full luminosity. Charm and other calibration and monitoring events will be taken and pre-scaled to keep the output rate below 2.5 KHz.

11.6.3 Risk Mitigation for L3 Processing Speed and Event Size Reduction

Two significant unknowns for the L2/3 Trigger system are the L3 Trigger processing time and the L3 output event size. These two are correlated since the faster the L3 trigger processing is done the more likely we will not need raw data and can summarize (reduce) the L3 output data into a smaller size that does not include as much raw data, only physics quality data. The uncertainty is caused by the lack of a full L3 trigger code and the uncertainty in the CPU performance at the time when the L2/3 Farm worker PC's are bought. The mitigation of this risk in terms of the amount of L3 processing that can be done and the event size out of L3 is described in this section.

It is reasonable to assume that attention to code speed can be achieved since the traditionally offline reconstruction code for BTeV is all developed as part of the L3 Trigger software project, rather than a much more open-ended offline project. The ultimate aim is that the efficiency and rejection goals of the L3 Trigger are met and the processing take no longer than 134 ms/event. The rejection requirement for L3 is only a factor of 2 reduction compared to the L2 rate. There is flexibility in optimizing the rejection at each stage of the trigger, so that if necessary either L1 or L2 could provide extra rejection. In addition there is considerable scope contingency in the L3 trigger since not everything included in a full "offline-like" production need be run at L3.

Another point to note is that the L2/3 PC farm is scalable. As part of the scalability requirement, additional L2/3 processing PC's can be added to increase the capacity of the L2/3 PC farm.

Mitigation for Slow L3 Processing

Although the full "offline" reconstruction code is developed and written as part of the L3 Trigger software project, we would not necessarily need to run all of it actually at the L3 Trigger stage. Some of the components of the full "offline-like" processing could be deferred till after L3. This additional processing can be done on free cycles of the L2/3 PC farm, since we assume a duty cycle of 33% for beam (when averaged over long periods), or it could be done at PC farms at collaborating universities. As the code matures and is optimized for speed, more and more of the full "offline" reconstruction would actually be run at the L3 Trigger stage. Some offline processing would probably always be desirable, like splitting off different physics data streams, or adding additional interesting physics streams.

We have already shown in section 11.6.1.1 that the L3 Trigger requirements would very likely be met by performing a part of the full event reconstruction (charged particle tracking). Nevertheless we describe a scenario to handle the possibility that the L3 processing is too slow. In all scenarios we would try to not lose interesting physics data. The results of simulations given in Table 11.6 show that the L2 Trigger can be run well within existing CPUs. We thus assume that the L2 trigger is always run and consider scenarios where L3 is not run at all, where the full L3 is run on a fraction of the events, or where partial L3 is run on all events.

The BTeV Level 2/3 trigger calculations are performed by a farm of 1536 CPUs, which are split into eight Highways. Each highway receives up to 6250 events per second that have passed the Level 1 trigger. Each of the 192 processors of a highway receives about 33 events per second and performs the Level 2 calculation on all of them, using on average 5 milliseconds/event. Only 10%, or about 3.3 events, pass the Level 2 selection. The 3.3 events are then passed to the Level 3 calculation that uses an average of 134 milliseconds. The Level 3 trigger passes about 50% of the events, which comes to 1.6 events/CPU per second, or about 313 events per highway per second.

In the worse possible case of not running the L3 Trigger at all we would run the Level 2 software on all the events, taking 16.5% of the time on each CPU. On average, 3.3 events/sec per CPU would pass the Level 2 trigger and require processing through Level 3 on the same processor. If no Level 3 processing were done at all the output could be stored on local disk for 67 hours (2.8 days), however this is not enough buffering to reduce the total instantaneous output rate out of L2 of 1200 MB/s to a manageable level. Some additional rejection would have to be provided by retuning the L1 and/or L2 trigger selections. For example, in older studies we found that the L1 trigger could be run to get a 99% rejection of min-bias events with only a small difference in B signal efficiency loss. In addition the L2 trigger could still get a rejection factor of 10:1 for the events that pass L1 [3].

More realistically, instead of no L3 processing at all, a fraction of these events could be processed through the L3 Trigger while the rest could be written to local disk for later processing. With 200 GB of disk per CPU, the remainder of the events could be stored for later processing - either during non-data-taking periods or sent to an off-site processor farm at a collaborating institution. (Note that in fact one does not need to wait for non-data-taking periods, as additional CPU cycles will become available during the store as the luminosity decreases.) As an example, if the L3 processing took 268 ms instead of 134 ms, we could process 1.6 events/sec through L3 with an output of 0.8 events/sec. The other 1.6 events/sec could be stored on local disk for as long as 136 hours (5.7 days). If there is not enough buffering to process all these stored events through L3 during more idle or non-data-taking periods, they would have to be processed through L3 by other computing resources, e.g. those located at the universities of collaborating institutions. Note that the data rate out of L2 is 10 times smaller than the incoming rate and can be easily handled by the DAQ network if needed.

An alternative to not doing the L3 processing, or doing the full L3 processing on a subset of the data is to do partial L3 processing on all events (crossings). The results of simulations given in Table 11.7 show that the full charged particle tracking should be able to be performed within the maximum allowed L3 processing time. This means that the partial L3 processing could consist of the full charged particle track reconstruction for which some event rejection could be done. Moreover since the raw pixel data makes up 50% of the raw event size, some data reduction can also be achieved by dropping raw tracking detector hits. Again use would be made of the local disk on each CPU to store this data output from partial L3 processing for a substantial time. As an example, if a rejection factor of say 1.5 (instead of the normal 2) could be achieved with partial L3 processing, the L3 output data could be stored for about 100 hours (4 days) if no data reduction is done. With a data reduction factor of say 2 (instead of the normal 3), this storage time would be extended to 200 hours, or about 8 days. This should provide enough time to perform the rest of the L3 processing either on the L2/3 Farm during more idle cycles or on off-site farms at collaborating institutions. It should be noted that in section 11.6.1.1 we estimate that even with just this partial L3 processing (consisting of the full charged particle track reconstruction) we would very likely achieve our L3 processing latency, background rejection and data reduction goals.

Mitigation for Larger than Expected L3 Output Event Size

The other uncertainty is whether we can obtain the specified 3 times reduction in event size to 80 KB/crossing. This specification is due to the requirement of writing output at less than 200 MB/sec to the data archival system. This requirement is to keep the archival storage at a reasonable level (about 2 petaByte/year) and is not a technical data rate limit. We have already mentioned above how, with partial L3 processing and partial data reduction the data could be stored locally on each Farm Worker node, so that further processing could be done during more idle or non-data-taking periods, or on an off-site farm. In the worse case with no L3 processing at all, some adjustment of the L1 and L2 rejection rates would have to be done. We consider a more realistic case where partial or full L3 trigger processing is performed on all events, but the full data reduction factor of 3 is not achieved.

If the full L3 processing is done and only the data reduction was not done, i.e. keeping 250 KB/event then the instantaneous output rate would be 625 MB/sec. (With enough buffering this rate could be reduced to 208 MB/sec with the assumed 33% data taking duty cycle. However there is only enough buffering for about one week which is probably too short to assume a data taking duty cycle of only 33%.) If no data reduction could be done at all then more data storage would be needed to create a long enough buffer. The situation is slightly better than stated as we have assumed the event size to be 250 KB out of L3 when no data reduction is done. In fact the raw pixel data contain more than 3 bytes of time stamp information per hit which can be eliminated once the event is assembled.[4] So even with the information added at the L1 and L2 stages the estimated event size would be more like 208 KB/event, or 156 KB/event with a 75% data compression (e.g. using gzip).

It is likely that with full, or even partial L3 processing, some data reduction can be done. For example the results for preliminary L3 tracking (see Table 11.7) shows that the

[4] Although this could be done at an earlier stage of the trigger than L2/3, we have not assumed this for the baseline at this time.

charged particle tracking should be able to be done at L3. With full charged particle tracking done one could eliminate the raw data from the pixel, straw and forward silicon detectors. Even with just pixel-only tracking done the event size would be significantly smaller by eliminating the raw pixel data. The estimated event sizes (including headroom as explained in Sec. 11.6.1.1) are given in Table 11.11. Note that the options listed in Table 11.11 are just some of options that could be used for data reduction. Other options can include for example keeping some of the raw pixel or tracking hits that are associated with particular tracks, or keeping some hit clusters but dropping raw hits. The options will become clearer as the L3 software projects evolve.

Even though it is likely that the data reduction could be achieved with just the charged particle tracking at L3, in a similar manner to the previous section, we have considered different scenarios where the entire L3 trigger rate rejection or data reduction is not achieved. We give some mitigation actions that would be needed, but they should be taken as examples of the options that would be available. Any actual mitigation choices would warrant careful consideration depending on the L3 software progress and the actual running conditions. Table 11.13 summarizes some different scenarios together with examples of the mitigation actions that might be needed. It should be noted that a simple alternative is to just write out more data at L3 which would require more archival storage.

Condition	Instantaneous Rates	Buffered Rate	Buffering Available	Actions
Normal: rej. ×2 data red. ×3	200 MB/s 80KB/BCO 2500 evt/s	67 MB/s	426 hrs	∼1KHz b-quark ∼1KHz c-quark + min.bias ∼0.5KHz calibration
No L3: rej. ×1 data red. ×1	1250 MB/s 250KB/BCO 5000 evt/s	417 MB/s	68 hrs	Adjust L1, L2 rejection Add more DAQ disk buffering + send some data off-site
No L3: rej. ×1 data red. ×1.2 no time stamp/pixel hit	1040 MB/s 208KB/BCO 5000 evt/s	347 MB/s	82 hrs	Adjust L1, L2 rejection Add more DAQ disk buffering + send some data off-site
Partial L3: rej. ×1 data red. ×1.8 pixel tracking	700 MB/s 140KB/BCO 5000 evt/s	233 MB/s	121 hrs	Add more DAQ disk buffering + send some data off-site
Partial L3: rej. ×1.5 data red. ×1.8 pixel tracking	467 MB/s 140KB/BCO 3333 evt/s	156 MB/s	182 hrs	Use DAQ disk buffering
Partial L3: rej. ×1.5 data red. ×2.5 Full tracking	333 MB/s 100KB/BCO 3333 evt/s	111 MB/s	256 hrs	Use DAQ disk buffering
Full L3: rej. ×2 data red. ×1.2 with all raw hits	520 MB/s 208KB/BCO 2500 evt/s	173 MB/s	164 hrs	Add more DAQ disk buffering + send some data off-site
Full L3: rej. ×2 data red. ×1.8 without raw pixel data	350 MB/s 140KB/BCO 2500 evt/s	117 MB/s	243 hrs	Add more DAQ disk buffering
Full L3: rej. ×2 data red. ×2.5 without raw track hits	250 MB/s 100KB/BCO 2500 evt/s	83 MB/s	341 hrs	Use local disk buffer

Table 11.13: Summary of some example L3 scenarios. Under the conditions column, the achieved L3 data rate rejection factor and the data reduction factor are given. Given in other columns are the rates, event sizes and output data rates and any actions needed. The (fully) buffered rate assumed there is enough disk buffer so that one can use an averaged 33% data taking duty cycle. The hours of buffering available listed includes only the disk buffering provided by local disks on the farm worker PC's, the DAQ disk farm could provide additional buffering. The buffering time gives the available time for either reducing the instantaneous data rate to a lower average data rate and/or for offline processing to reduce the archival data storage needed. Note that data rates could be further reduced with data compression (e.g. gzip), and that more options are available for data reduction and mitigation than just the examples listed here.

11.7 Trigger Supervision and Monitoring

11.7.1 Overview

The Trigger Supervisor and Monitor (TSM) is the system that resets, initializes, controls, and monitors status and statistics from all component modules of the BTeV trigger. The TSM system is self sufficient in that it provides complete control and monitoring capabilities for the trigger without relying on external systems for data transport. Any products that are used to construct the TSM are considered part of the TSM system. Furthermore, it encompasses a hardware platform that is used by RTES [28] to implement fault detection, fault mitigation, and error handling functions. The TSM includes fault detection and mitigation infrastructure. The RTES Collaboration is developing fault detection and mitigation software that will be used as optional components with the TSM infrastructure.

The BTeV trigger consists of a large collection of distributed processing elements operating on detector data that flows in parallel through multiple data streams. Fig. 11.35 shows a block diagram of the trigger system showing some of the parallelism (specifically the trigger highways) as well as the interconnections between the data-processing hardware, the DAQ, and the TSM. The data-processing elements range from "hardwired" logic elements to powerful data-processing computers with varying capabilities for receiving, acting upon, and generating messages that are used to implement necessary supervision and monitoring capabilities. The supervision and monitoring of so many elements is a technical and operational challenge. The TSM system is critical for maintaining adequate resources during BTeV data taking, and the system itself is complex enough so that some of its resources will be needed to maintain its own operational state. This will be accomplished with the aid of fault detection and fault mitigation tools provided by RTES.

11.7.1.1 Functional requirements

The TSM system provides the connectivity to send and receive messages or stream data containing commands, initialization sequences, configuration data, error messages and data pertaining to status and statistics for subsystems in the BTeV trigger. Functional descriptions of these messages are classified as supervisor functions or as monitor functions.

Supervisor functions

- initialization and configuration
- command execution and distribution to subsystems
- autonomous error handling

The TSM configures both the trigger hardware and software. The initialization is hardware dependent. The initialization of L1 trigger subsystems, for instance, includes downloadable FPGA firmware, execution code for processors, set points, and various trigger tables.

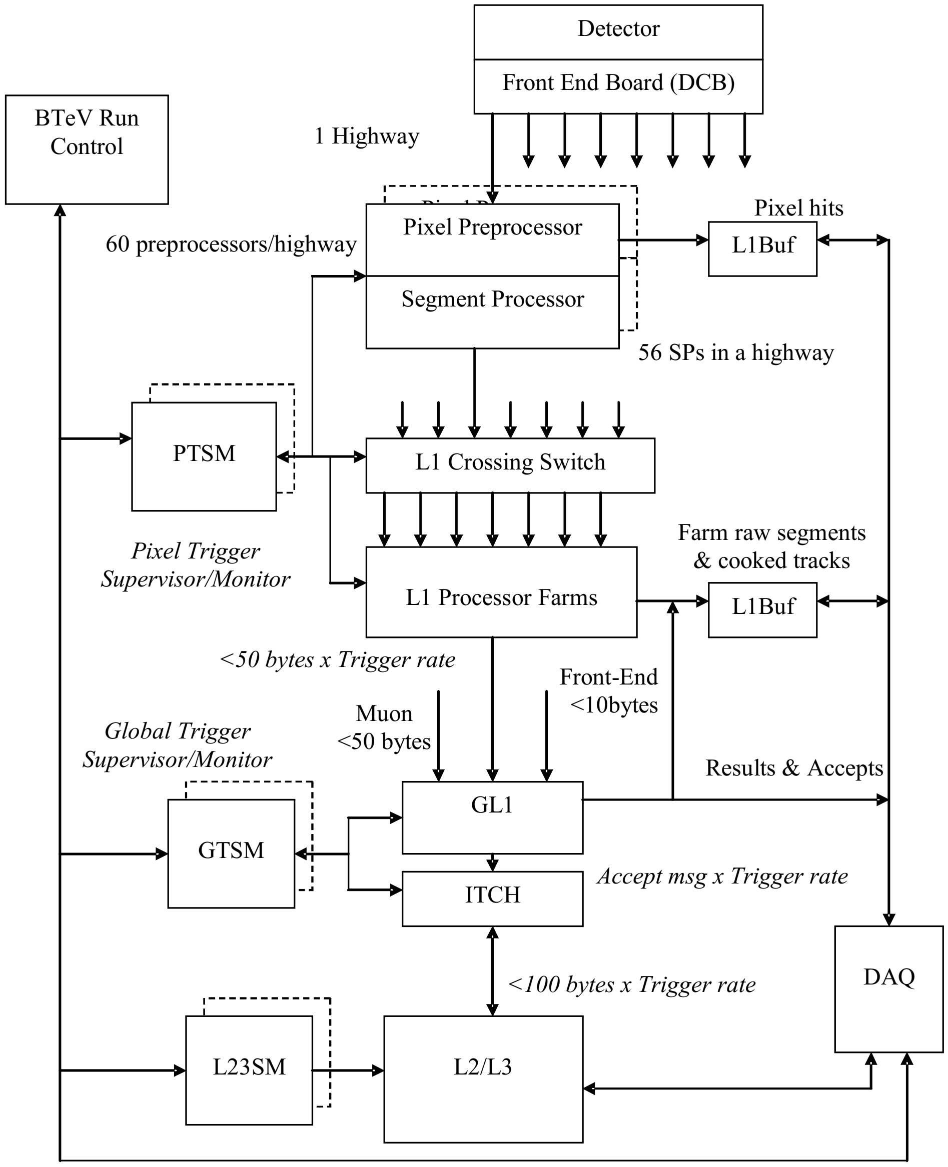

Figure 11.35: Block diagram of the BTeV trigger and interconnections between the data-processing hardware, the DAQ, and the TSM system.

Initializing the entire trigger system is a large task that may take several minutes, but we expect that a complete initialization of the system will not happen often. As described in the architecture section below (see Section 11.7.2), the TSM system has a hierarchical structure. The intermediate and lower level nodes in this structure have enough intelligence and storage resources to start up data processing and supervisory functions locally. During commissioning, these nodes will be updated frequently as changes and improvements are propagated through the system. After the trigger has been commissioned, the initialization of the processing nodes and startup of the system will take less time. During normal running, code updates to the intermediate nodes can occur in parallel with data taking. In this manner the system will be streamlined as it is integrated with the BTeV detector and electronics.

TSM messages are, for the most part, organized by the physical arrangement of the hardware. High level command messages will be translated, as needed, at each TSM intermediate node so that messages can be propagated to low-level nodes, which may have varying capabilities for supervision and monitoring. Furthermore, there will be different broadcast attributes so that commands can go to a single hardware component, a subset of components, or an entire collection of components.

Monitor functions

- error message collection and reporting
- hardware and software status messages
- status message collection and logging
- data histogramming

The TSM reads data from all programmable devices in the three trigger levels, L1, L2 and L3. This includes the programs, parameters, device configurations, status and error messages, temperature and voltage measurements, as well as processed data at useful probe points in the data stream.

The characteristics of the messages used to monitor trigger components are the following. Since the bandwidth of the TSM network may be limited compared to the amount of monitoring data that will be available, each level of the TSM hierarchy will have the resources to perform data “compression.” For example, as monitor messages propagate up from low-level hardware to the intermediate TSM nodes, the amount of data can be reduced by having the intermediate node send a single “group reply” that indicates that all of the low-level hardware components reporting to the node have replied to a request. The extent of this kind of data compression will be controlled by operational policy, where a policy is a coherent set of parameters that activate and configure “pluggable” software components. The policy or its parameters can be changed dynamically depending on running conditions. Hence, for debugging and commissioning purposes, the compression may be completely disabled so that every message is passed from the lowest levels to the highest. For normal operations, compression may be engaged at a “normal” level (determined by policy). Whereas a response to persistent error conditions, could cause the compression to be enhanced by exponential prescaling (for example, send each of the first 5 messages, then a message of the form “5 more of,” then a message of the form “25 more of,” then “125 more of,” and so on).

Another form of “compression” will be applied to histograms that are generated for monitoring purposes. Similar to the “group reply” that is generated by an intermediate node in response to a request, individual histograms from low-level worker nodes can be combined (for example, by adding all of the individual histograms together) to reduce the volume of data sent over the TSM network.

11.7.2 Architecture

The TSM system and its network can be viewed as a pyramid structure with a host node at the top, and a message distribution network that expands through intermediate nodes down to worker nodes at the bottom. A block diagram of the TSM system is shown in Fig. 11.36. At the top of the figure is the TSM Host (TSMH) that communicates with subsystem-specific TSM hosts, which in turn communicate with TSM workers. The TSM subsystems are the following:

- PTSM - pixel trigger supervisor and monitor
- MTSM - muon trigger supervisor and monitor
- GL1SM - Global L1 trigger supervisor and monitor
- L23SM - L2/3 trigger supervisor and monitor

These subsystems implement the connection to the trigger processing elements and can be implemented as relatively powerful processor nodes, as software threads in the target trigger processor element, or as dedicated hardware connected to the trigger elements. The flow of information will be almost exclusively vertical, with commands flowing from hosts to workers and error and status messages flowing back to the host. The TSMH is controlled by BTeV run control during data collection or detector commissioning, but the host can also perform diagnostics and housekeeping tasks in parallel when resources are available. The subsystem-specific TSM hosts are fully functional as the logical root of their TSM tree. The PTSM host, for example, can control the entire pixel trigger for debugging and commissioning purposes. The TSMH serves as the master control for subsystem-specific TSM hosts, and can also imitate run control functions during commissioning.

The TSM system will provide the connectivity for all of the above described messages. The system must provide extremely reliable delivery of these messages. However, the latency requirement of commands is mitigated by the fact that all data streams are tagged with the crossing number and do not need to be tightly synchronized. Furthermore, the detector data flowing through the DAQ and trigger is not synchronized across the data paths, but it is

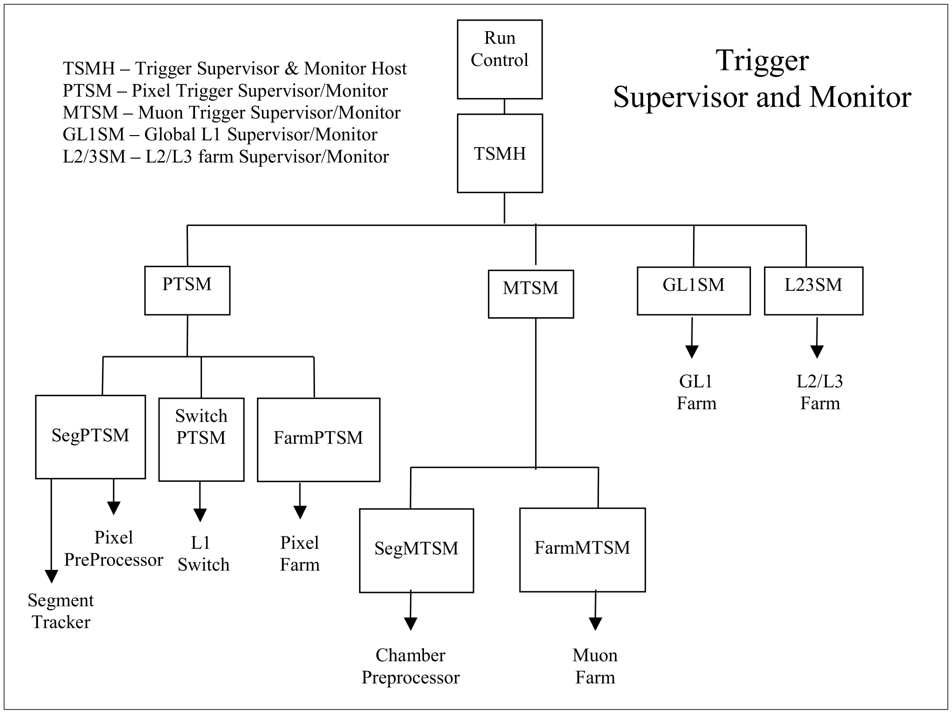
Run Control
TSMH – Trigger Supervisor & Monitor Host
PTSM – Pixel Trigger Supervisor/Monitor
MTSM – Muon Trigger Supervisor/Monitor
GL1SM – Global L1 Supervisor/Monitor
L2/3SM – L2/L3 farm Supervisor/Monitor
Trigger Supervisor and Monitor
TSMH
PTSM
MTSM
GL1SM
L23SM
GL1 Farm
L2/L3 Farm
SegPTSM
Switch PTSM
FarmPTSM
Pixel PreProcessor
L1 Switch
Pixel Farm
Segment Tracker
SegMTSM
FarmMTSM
Chamber Preprocessor
Muon Farm

Figure 11.36: Block diagram of the Trigger Supervisor and Monitor system.

locally synchronized within the processing elements whenever necessary. This design feature allows relaxed latency requirements for the entire trigger system and DAQ at the cost of larger data buffers in the data paths. Recent decreases in the price of memory supports this design decision. The message latency requirement for the trigger is not stringent and is therefore not a significant driver for the hardware design. Most current state-of-the-art data link designs can meet the link rates and bandwidth needed. Errors will generate messages to be sent to control elements but fault mitigation will be attempted as close as possible to the hardware reporting the error. The error messages can take some time to get to the host, but it must be reliable delivery

The determination of link bandwidths in the TSM system requires a compromise between a fairly low rate needed for normal operations consisting mostly of traffic pertaining to command and status messages, and the infrequent need for high rates to shorten the time needed to download FPGA code, trigger tables, and application code or to accommodate the error and recovery messages of a large failure. The peak link bandwidth will be determined by the bandwidth needed to start up the trigger system in a "cold start" mode, where some reasonable amount of initialization and configuration information needs to be distributed to the trigger processor elements. In a "warm start," after the system has run and halted, startup can be very quick due to reuse of much of the configuration. During normal operation, some of the TSM bandwidth will support the movement of histograms used to monitor the trigger operations.

11.7.3 TSM Software

The heterogeneous architecture of the TSM system requires a diverse collection of software elements. The TSM software will be described first in terms of the required elements that are common to all levels, then in terms of specific differences between levels. Finally, the relationship between the TSM software and the RTES project [28] is described.

TSM Similarities

Each processor of the TSM system must be capable of at least a minimal self-boot on power up, such that human intervention is not required to deliver the processor to a state where it can receive messages from a higher level. Message handling resources must be provided for down-coming, up-coming, down-going, and up-going messages. Since the TSM is a hierarchy of processing resources, command and control messages will come down from above (down-coming), and after local processing, if appropriate, will be distributed to lower level processors (down-going). Responses and errors may be received from lower levels (up-coming), and after local processing, if appropriate, will be forwarded to the next higher level (up-going). Interrupt service and task scheduling, at least at a primitive level, must be supported.

At each level, the capabilities of the TSM processors will be used in accordance with established policies. Policies correspond to configuration-defined controls that enable or disable distinct capabilities on a particular processor. For example, any TSM processor will be capable of autonomous down-going and up-going traffic as a result of local processing, which results in either locally generated command and control to the lower levels (automated recovery), or locally generated status and error reports to a higher level (self-initiated detection). However, these features will not be active unless allowed by downloaded policy configuration data. Another example concerns network bandwidth. Mid-level TSM processors will be capable of accumulating messages and providing condensed summaries (for example, all Farm Worker Nodes have booted successfully), as well as forwarding individual messages (Node N has booted, Node N+1 has booted, etc.). The extent of data compression (as mentioned in Section 11.7.1) will be determined by policy configuration data. This capability/policy approach will allow debugging and commissioning to have access to as much or as little knowledge and control of the system as required from the top level, and it will allow normal operation to be tuned for performance by trading autonomy for detail.

Finally, each processor of the TSM must be able, as appropriate, to support code and configuration caching. As part of the self-boot process, or in response to a command from a higher level, each TSM processor will attempt to load a local copy of its executable code, and will attempt to configure itself based on a local copy of its configuration data. An important first step in this process, however, is to certify that the local copy is valid. To do this, each TSM processor will review its local storage resources, compute checksums, and compare them to checksums provided by the next higher level of the TSM system. If the local copy is valid, it is loaded without further intervention. If the local copy is not valid either by virtue of being obsolete or corrupt (i.e. the checksums do not match), the TSM processor will wait for the next higher level processor to provide a valid copy of the code and/or configuration data. Symmetrically, each TSM must, as appropriate, service requests for code and configuration coming from below, either from its own storage resources (if the checksums match), or by forwarding the request to a higher level.

TSM Differences

The TSM processor hierarchy consists of Linux processors (with local disk storage) or Linux processes with access to disk storage at the highest levels, and diskless embedded processors (microprocessors, DSP's, or FPGA's with embedded processor cores) at the lowest levels, with a range of processing and storage resources in between. In particular, the lowest level consists of a local manager process, within an embedded worker processor. At each level, the resources dictate slightly different capabilities and behaviors.

At the topmost TSM level, the TSM host (TSMH) must be able to self-boot, and must be able to automatically start all services and initialize all required resources. We envision the TSM host to be a Linux process, or processor, with startup scripts that are invoked during the boot process. Message handling will include message archiving to a BTeV-standard database. Also, a graphical user interface will be native to the TSM host to display system status and to allow both global and selective command and control. However, in normal operation, the TSM host will defer to BTeV Run Control for primary start, stop, and report commands. The TSM host need not necessarily run a real-time operating system.

At the subsystem-specific TSM host level, most of the TSMH features will be available, but many will be disabled. To support debugging and commissioning, each subsystem-specific TSM host must be fully capable of acting as if it were the TSMH, for a reduced implementation of BTeV involving only that subsystem. Hence the pixel trigger can be fully managed from the PTSM host, and the muon trigger can be fully managed from the MTSM host. This includes messaging and the user interface, but may exclude message archiving. During normal operation, the subsystem-specific TSM hosts would defer to the TSMH for all command and control, acting largely as an intelligent message passing element. This intelligence would be manifest by converting generic start, stop, and report commands into subsystem-specific equivalent commands, as well as interpreting status and error messages from lower level processors and composing higher level summary reports. The subsystem-specific TSM hosts need not necessarily run a real-time operating system.

At the regional TSM processor level, of which there may be more than one depending on networking considerations, mid-grade computers, with or without disk, will be deployed. The primary role of the regional TSM processors is to provide message handling, and as controlled by policy configuration, mid-level error detection and mitigation.

At the Farmlet manager level, an embedded processor will be used. Its resources will be severely constrained. It may be diskless, but will have flash memory to act as a storage resource. In addition to message handling between the regional TSM processors and the embedded worker processors, the Farmlet manager will also be responsible for direct, low-level control of the workers.

The lowest level of the TSM hierarchy is the local manager level, implemented as code within the embedded worker processor, along with the physics application that performs the fundamental function of the trigger. The local manager process will accept messages from the physics application (via API calls), and will forward them to the Farmlet manager. The local manager, as directed by the Farmlet manager, may stop and/or reload the physics application, or change its configuration variables. The local manager will operate under, or integrated with, a real-time kernel serving the embedded worker node.

As described above, each level of the TSM must be capable of playing an appropriate role in code and configuration caching. The topmost TSM host will be able to directly obtain code (from a BTeV standard database or from a local cache) necessary to operate without human or ancillary system intervention. The lowest level TSM process, or processor, is relieved of the obligation to provide code or configuration to lower levels, as it is at the bottom level. Furthermore, the lowest level TSM process is exempt from having a code or configuration cache; it is free to act as if the "local copy" (which does not exist) is always corrupt, and must always be downloaded from the next level up.

L23SM Subsystem

While the preceding discussion has been focused on the L1 trigger, the same perspectives apply to the L2/3 farm, with the following small differences. All TSM "processors" are processes, and each runs on a Linux computer. The "depth" of the regional TSM hierarchy may be very shallow, and there will be no layer that corresponds to the "farmlet manager" layer in the L1 trigger.

TSM software, and RTES

The TSM will most likely need to utilize external products (libraries and executables). Only those products that pass BTeV quality standards, have undergone sufficient testing, and can be supported by the staff (administrative and software development) will be considered. Any products that have value, but have not passed all the criteria for inclusion, will be treated as add-ons or plugins, which can be easily excluded. This includes RTES components. The TSM will have a basic, limited operating mode that will allow the system to be booted, verified, and run with a minimal set of core external products. The TSM will define the abstractions or APIs and component architecture, which allow other systems or products to add new functionality to it without producing a new software release. RTES and other groups will be required to build components to this interface. The RTES group will work closely with BTeV to develop their systems to the same standards to minimize research and development time up front, and maintenance costs in the end. Areas of shared interest are most likely the message/data passing infrastructure (protocols and formats) for commands and monitoring information, database interactions, sensor reading, and APIs used by BTeV developers.

11.8 RTES

11.8.1 Overview

The BTeV trigger system encompasses several thousand CPUs distributed over a three-level trigger architecture. Time critical event-filtering algorithms that run on the trigger farms are likely to suffer from a large number of failures within the software and hardware systems. If fault conditions are not given proper treatment, it is conceivable that the trigger system will experience failures at a high enough rate to have a negative impact on its effectiveness. It is also likely that an administrative staff and cast of experiment operators will not be able to service simple problems, or analyze complex problems or relationships in a timely fashion to avoid data loss. This was called out in a technical review of the trigger system as an area of significant concern.

To address this concern, we have established the BTeV Real Time Embedded System Collaboration (RTES) [28]. Funded by a $5M grant through the NSF ITR program, RTES is a group of physicists, engineers, and computer scientists working to address the problem of reliability in large-scale clusters with real-time constraints, such as the BTeV trigger. RTES is defining a software infrastructure to detect, diagnose, and recover from errors not only at the system administration level, but also at the application level. This infrastructure must be highly scalable to minimize bottlenecks or single points of failure. It has to be verifiable to make sure that it performs its functions in a timely fashion, extensible by users to acquire new detection/analysis methods, and dynamically changeable so that it can be reconfigured as the system operates. The problem is being approached by using a hierarchy of monitoring and control elements. The architecture is such that lower levels have high data rates, short reaction times, and a narrow view of system components, while higher levels have aggregated data summaries, longer reaction times, and a more global perspective of the system.

The goal of the RTES project is to create fault handling that can be used by all components in the BTeV trigger and DAQ. This subsystem must be capable of accurately identifying problems and compensating for them. This includes application related activities such as changing algorithm thresholds, and overall system activities such as load balancing. As many recovery procedures as possible must be automated. A simple example is the ability of the system to switch to a hot-spare L1 processing board when a failure is detected in a board that is actively processing data. Operators and system developers must be able to easily incorporate new procedures or policies into the system. The operators must be able to easily select error handling policies, and a detailed record of observations and actions must be kept to facilitate reproduction of analysis results and to identify long-term trends.

Creating a single subsystem for handling faults across the DAQ and the trigger can benefit the experiment by lowering new procedure integration costs, and by reducing the amount of knowledge necessary to operate and maintain the system. This is done by introducing a standard set of interfaces and protocols that reduce the number of conversions and products that must be developed and maintained. The development of standards for error handling

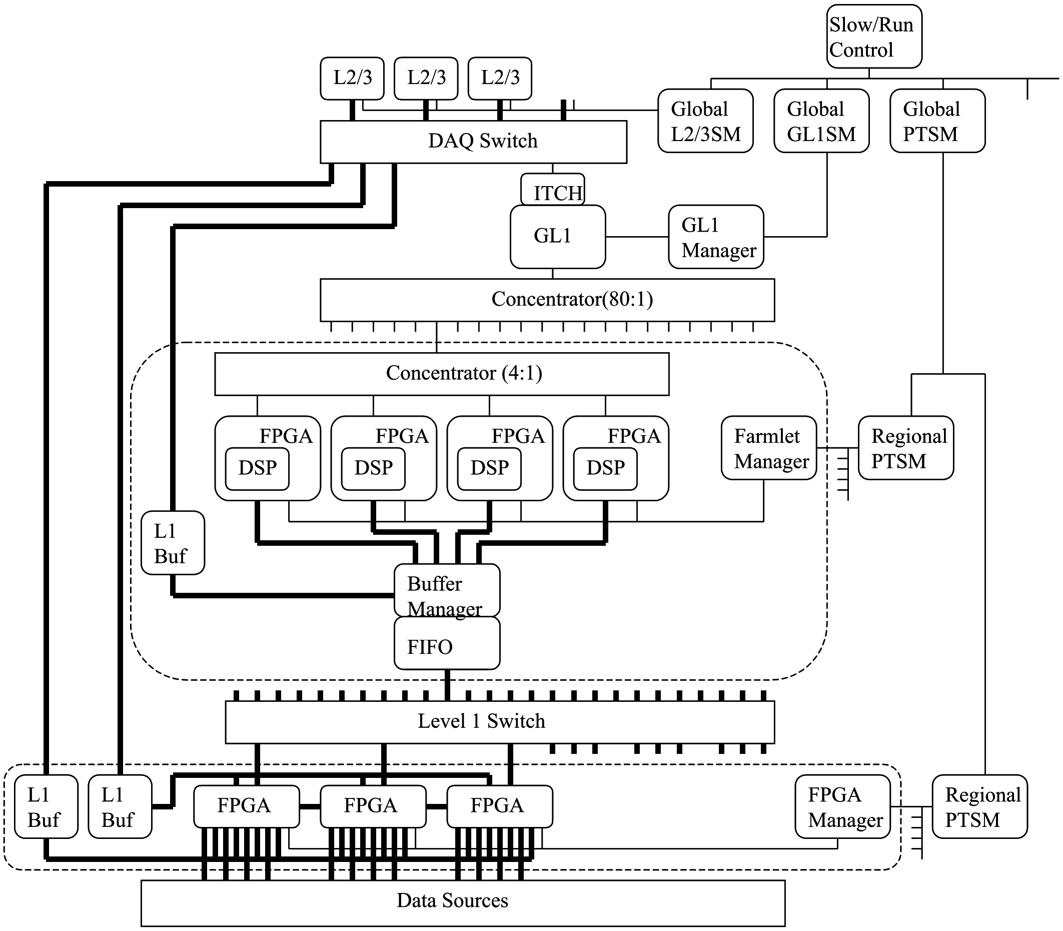

Figure 11.37: Block diagram of the SC2003 RTES model of the BTeV trigger components

and reporting means that the information produced or exchanged between applications can be easily processed.

Each of the universities involved in the RTES Collaboration has expertise in some aspect of the problem. In some cases, they already have hardware and software toolkits that have been used to solve smaller scale problems related to real-time embedded systems and fault management. In order to develop an embedded processor system to study issues associated with the Level 1 Trigger, RTES adopted a DSP-based prototype architecture for the trigger. This prototype helped to establish subsystem boundaries, gave a sense of scale, and helped identify required interfaces and error conditions. The prototype leveraged Vanderbilt University's expertise in DSPs as the embedded L1 processors. Fig. 11.37 shows a block diagram of the trigger components. The farmlet shown in the figure is essentially a single-event input queue (FPGA) with three to six servers. It also contains a microcontroller that is used for configuration, supervision, and monitoring.

The RTES Collaboration is currently engaged in developing a second prototype system using commodity processors, to understand the issues associated with the L2/3 Trigger.

Additional information on both of these research and development activities is presented below.

11.8.2 RTES Deliverables

The technologies introduced by RTES are discussed below. They are *ARMOR*s for L2/3 processors and Manager-I/O Host PCs, *VLA*s for embedded processors and specific monitoring tasks at L2/3, and *GME* for system modeling and configuration. These are the deliverables of the project. Each of these deliverables is used for different aspects of the trigger, and all of them must work together.

The deliverables form a toolkit. This includes core infrastructure (libraries and APIs) as well as methodology for organizing and creating fault tolerance components. The toolkit is designed to be expanded as the experiment comes to life. The toolkit will include many BTeV-specific fault detection/action components for both hardware and application software.

11.8.2.1 ARMORs

The University of Illinois has produced a fault management software component called Adaptive, Reconfigurable, and Mobile Objects for Reliability (ARMOR). ARMORs are multithreaded processes composed of replaceable building blocks called *Elements*. Elements communicate by way of messages. The architecture makes this a highly flexible system that includes modules such as recovery-action elements, error-analysis elements, and problem-detection elements that are developed and configured independently. ARMORs can be configured in a hierarchy across multiple processors to provide complete system coverage. Fig. 11.38 illustrates a simple armor configuration, where a node has a main ARMOR daemon watching over the node and reporting to higher-level ARMORs out on the network. Elements within these node-level ARMORs work together to make sure all nodes are operating properly. Another example of a standard ARMOR is the execution ARMOR. This ARMOR is responsible for protecting a single application. This type of protection does not require modifications to the program; it simply watches a running program. It can restart the application, generate messages for other elements to analyze, or trigger recovery actions based on returns codes from the application.

Within the trigger, ARMORs can provide error detection and recovery services to the trigger application and any other process running on L2/3 nodes. They can also watch for hardware failures. ARMORs are designed to run under an operating system (such as Linux or Windows [31]) and are not well suited for embedded systems with limited memory and processing-time requirements, for which we are developing VLAs (see next section). Using the ARMOR API, a trigger application can report specific errors as well as other information directly to one or more elements. For example, data processing rate measurements and data quality measurements can be sent directly to ARMORs to be distributed to running elements for analysis. The BTeV online group is currently evaluating ARMORs to watch over DAQ and trigger related processes [27].

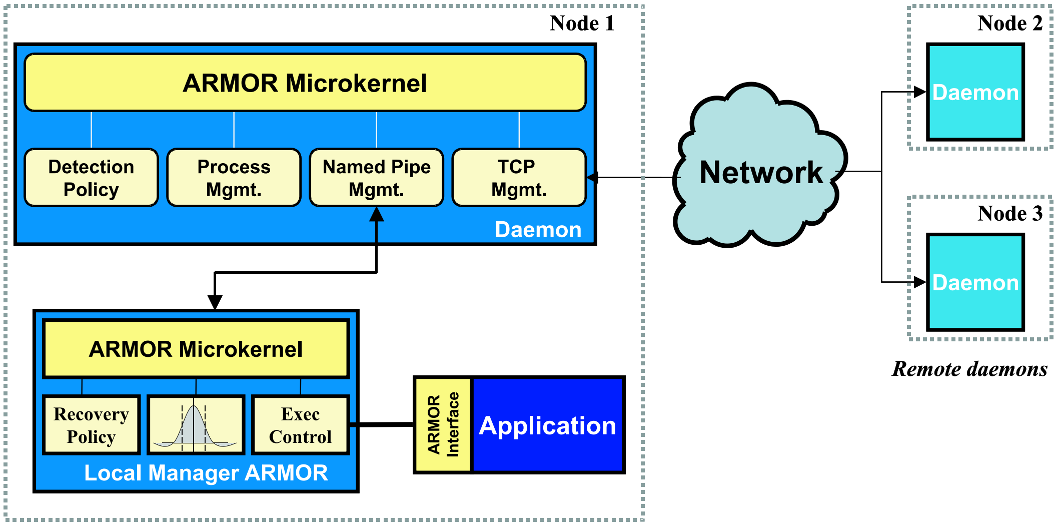

Figure 11.38: Simple ARMOR configuration

11.8.2.2 VLAs

Syracuse University and the University of Pittsburgh are developing a concept called the Very Lightweight Agent (VLA). A VLA is a software entity designed to collect various environmental and process related measures, analyze them, and perform actions in a highly constrained environment such as the L1 trigger. VLAs may be developed as standalone processes, threads, or a collection of functions that maintain the state of a subsystem within a larger application [33]. Given memory, CPU time, and network bandwidth constraints, a VLA will decide how to organize itself to provide the best possible results. In order to achieve this goal, a VLA may make use of real-time scheduling, priority queuing, and hierarchical rules to guide its decisions.

Within the context of the L1 trigger, VLAs will watch for fault conditions such as trigger algorithm crashes, link failures, the inability of processors to keep up with data rates, and processes running longer than expected. Since the L1 trigger is so restricted in terms of its resources, VLAs will rely on a higher-level control system to perform complex analysis and decision-making. It is easy to believe that running any amount of VLA code during the first part of data taking will cause the processor to fall behind. Therefore, a VLA will need to be smart enough to change its behavior as data taking progresses, to know problem priorities, and to know the best time to report status and fault conditions to the overall control system.

For the L2/3 trigger, VLAs may reside directly inside the trigger executable and perform similar function as in the L1 trigger. They will also be used to collect other hardware specific information such as CPU temperature readings and fan speeds.

An initial VLA prototype has been implemented and its interaction with an ARMOR was successfully tested. This test took place over the Internet, to verify the robustness and the distributed nature of the VLAs and ARMORs.

11.8.2.3 Modeling Tools

The ISIS group at Vanderbilt University has produced a graphical modeling tool called the Generic Modeling Environment (GME) [25]. The tool enables the user to do "Model Integrated Computing" (MIC). This tool allows a designer to model many aspects of a system by creating diagrams. The concept is similar to CASE tools in that it captures component relationships and properties. It is different in that it is not tied to a particular modeling paradigm, such as UML [32]. The tool allows a designer to create a domain specific set of rules that define modeling components. In essence the designer creates a modeling paradigm specific to a particular project. GME allows one to independently capture different aspects of a system using shapes, properties, associations, and constraints specific to the project and then combine them to form a system image [24]. Just as a compiler forms a parse tree from a programming language and then processes the information in the tree to create machine specific assembly code, GME creates a set of data structures representing the information in the models and allows "model interpreters" to generate information about the system. Typical model interpreters are C/C++ code generators and system configuration generators. Examples of aspects are hardware configuration, process dataflow, and fault handling. Hardware configuration includes physical components and their connectivity. Dataflow includes logical connectivity and executable configuration. Fault handling diagrams show system reactions to problems using hierarchical state machines. The look and feel of the GME is similar to electronic circuit design tools.

11.8.3 R&D Activities

11.8.3.1 Summary of R&D Activities

This section provides a brief summary of R&D activities that have been pursued by the RTES Collaboration.

ARMOR Support

A small farm of 6 Linux processors was configured to provide ARMOR support for run control, a database server, and two message services (run messages and error messages), as well as two client application nodes. This work was presented at the 13th Real Time Conference in Montreal in May 2003 [27]. Various failures and automated recovery were demonstrated. Although this system was developed for a "DAQ" application (*i.e.* run control), the exercise has been extremely valuable in coupling the ARMOR concept with trigger algorithm software. Further development, including ARMOR support for the Pixel Trigger and Muon Trigger Supervisor Monitor (PTSM and MTSM) hosts, will continue.

L1 Trigger Fault Detection and Recovery Support

A key requirement for any fault tolerance mechanism for L1 is rapid error detection so that no results are lost due to an error. Statistical integrity demands that every bunch crossing be accounted for. While error detection must be fast (at least as fast as the latency), recovery can be slower as long as the offending entity (hardware or software) is isolated from the rest of the system. Towards this end we have studied hardware and software resources provided by a TMS320C6711 DSP Starter Kit (DSK) from Texas Instruments, and the associated operating system (DSP/BIOS) from the same source [30]. This processor was chosen to be consistent with the RTES Demontration Project 2003 described below.

The DSK development environment (Code Composer Studio) and the Vanderbilt-developed DSP hardware are both developed to work with Microsoft Windows Operating Systems [31]. One of our accomplishments was the porting of ARMOR processes to Windows 2000. A formal model for reconfigurable ARMOR processes and a distributed, ARMOR-based, software implemented fault tolerance (SIFT) environment for managing user applications were presented in [34].

Furthermore, we have designed a detection/recovery scheme to handle race conditions, which could occur when the incoming data arrival rate exceeds the processing rate of a worker node. This is a somewhat hypothetical fault condition because buffer overflow (due to mismatched data and processing rates) would be handled by the Buffer Manager, not by the worker. However, this type of fault was easy to create, and provided a basis for exploring detection and recovery mechanisms. The detected race condition was reported to the host computer to initiate appropriate recovery actions. In order to improve our understanding of application and system behavior we explored the use of semaphores, blocked execution, software interrupts, hardware interrupts, and periodic tasks. Experiments were performed to obtain detailed knowledge of how software interrupts are handled inside DSP/BIOS, as well as determining limitations for the scheduling of multiple interrupts.

While these studies have been specific to the TMS320C6x family of DSPs and to DSP/BIOS in particular, we are developing an understanding of the infrastructure needs of BTeV and RTES code executing on a heterogeneous hierarchy of embedded processors, regardless of the actual processor(s) or kernel(s) used in the implementation of BTeV.

Finally, in an effort to provide fast hardware-level error detection we have explored the feasibility and potential implementation of a hardware framework for providing fault tolerance services. In this framework, error detection and recovery firmware and programmable hardware (registers, on-chip memory, and control logic) constitute a *reliability engine* implemented as an FPGA-based device, or fully integrated with the processor. The application can be instrumented (*e.g.*, using a dedicated pre-compiler) to instruct the processor about the desired level and type of runtime checking. The checking can range from full duplication of the instruction stream, to precise spot-checks of individual instructions and/or results produced by critical code sections. Several prototype timer elements have been conceived. Discussions are ongoing with regard to the implementation of these ideas in communications support FPGA(s).

L2/3 Trigger and Supervisor Host Fault Detection and Recovery Support

The L2/3 trigger consists of general-purpose processors (*e.g.*, Pentium, PowerPC), most likely running Linux. This trigger is responsible for a more sophisticated evaluation of the experimental data initially filtered by the L1 trigger.

In addition to the L2/3 trigger processors, there will be general-purpose processors responsible for system control functions. These "host" processors, or processes, include database service, time service, run control, slow control, and various supervisory elements. Some of the host processes or processors are part of the DAQ. Run Control and Slow Control are prime examples. However, to the extent that the hosts run on processors that are members of the L2/3 trigger farm, it is of considerable value to develop ARMOR support for the host applications. In the case of the Pixel Trigger and Muon Trigger Supervisor Monitor (PTSM and MTSM) hosts, ARMOR support is a specific deliverable of RTES.

The relationship between ARMORs and VLAs has been explored using LM_Sensors [26], a Linux software package for accessing low-level health monitoring hardware, commonly implemented in PCs built after 1997. It is likely that the processors of the L2/3 trigger will include such health monitoring hardware. Hence LM_Sensors are a convenient candidate for developing VLA concepts on a general-purpose processor. Using a standard messaging protocol developed by Pittsburgh for the client VLAs, a corresponding ARMOR server element has been developed. Sockets are used to abstract the relationship between the VLAs and the ARMOR. This is important, as it will allow the ARMOR recovery mechanism freedom to restart the VLA-server (ARMOR) on a different node in the event of a failure. Work is in progress to develop the managerial relationship between the ARMOR and VLA (starting, stopping, and naming VLAs), as well as both VLA failure recovery and ARMOR server failure recovery.

VLAs

The RTES collaboration has developed two VLA prototypes running on TI DSPs for L1. The first prototype was built for the TI DSP/BIOS operating system. The second prototype was developed for the hardware built by the Vanderbilt group running a proprietary kernel. A paper titled "Design of Very Lightweight Agents for Reactive Embedded Systems" was presented at the 10th IEEE International Conference on the Engineering of Computer Based Systems (ECBS).

We have also developed a scheduling algorithm, which takes advantage of the decreasing data rate (due to decreasing luminosity in the collider) during a *store*. We have tested the feasibility of this algorithm using a resource kernel on Linux with a simulated physics application and a VLA. A paper titled "A Real-Time Scheduler in a Large-Scale Embedded System" was submitted to EMSOFT.

Ongoing work with VLAs includes the following:

- Identify scenarios in the trigger systems pertinent to the design of VLAs
- Design and evaluate a task-scheduling scheme that takes advantage of the decreasing bunch-crossing rate

- Determine time requirements for the error handling routines of the VLA

- Emulate an embedded worker node on a Linux node such that the emulation includes a running VLA, a simulated trigger algorithm, and a command processor

- Study the performance of VLAs in an environment that includes various resource limitations

- Create a simulator to study the behavior of VLAs under different load conditions and different fault-injection assumptions

- Study the interactions between at least two levels of VLAs, with the frequency of communication between VLAs and ARMORs determined by communication delays

GME

The GME modeling tool is one of three deliverables provided by the RTES Collaboration. There has been considerable progress on the development of GME for BTeV system modeling, system generation, simulation modeling, and the BTeV runtime infrastructure.

The use of GME for system modeling encompasses several aspects that are *captured* by the modeling tool. The aspects that are currently included are the following:

- The hardware infrastructure for the BTeV trigger and DAQ including both processors and interconnections between processors and their attributes, which capture performance and fault probabilities.

- Tasks are captured, along with data paths and data dependencies between tasks. The task attributes specify the function that is performed by a task and its performance attributes. For example, tasks include physics applications, VLA diagnostics, and fault-manager tasks.

- State machines capture the desired fault mitigation strategies for BTeV. For these state machines, the states correspond to failure modes of the system, while transitions between states encompass conditions (such as logic equations based on local fault sensors, regional fault conditions, global faults, or user input) and actions. Actions specify the operations to be performed when the conditions are satisfied.

- An additional aspect has been included to model the composition of ARMOR elements based upon the particular category of an ARMOR, the location of the ARMOR, and the messages that are handled by the ARMOR. This information is used to generate scripts that instantiate and activate ARMORs.

In addition to system modeling, there has been progress on system generation tools that are used in both offline and online environments. To date, the focus has been on the offline (design-time) mode. The system generation tools process the system models and generate several products:

- System configuration for bootstrapping the network.

- System configuration files needed to begin execution. This includes, but is not limited to, creating and starting tasks, establishing connections, and routing data between processors.

- Fault mitigation modules for executing the behavior specified in the mitigation models. The modules consist of synthesized C code that monitors input variables, evaluates transition conditions, and maintains the state of the system. During transitions, the user-specified behavior is executed.

- Link rerouting tables are generated to perform rapid network reconfigurations for processor/link failures. Application-specific redundant signal paths are inserted into the software configuration, and scripts to manage the swapping of data paths are also generated.

The GME modeling tool is also applied to the runtime infrastructure. The development of this infrastructure includes the following:

- Support for link swapping: An API function has been implemented that utilizes the information in the rerouting tables and redundant signal paths to perform a swap in a communication link in the event of a link failure.

- Support for mitigation operations: Additional API functions have been implemented to support fault-mitigation operations. These API functions implement facilities for dispatching a fault message, resetting a faulty communication link, resetting a faulty processor, and relocating a process.

- Support for fault detection and fault mitigation messages: Data structures have been defined for propagating fault messages. These data structures have fields for message types, the severity of a failure, and the location of fault, to name a few.

Simulation modeling includes the creation of a detailed model of the system. The goal of the simulation is to generate accurate predictions of system behavior as well as the timing of the behavior for use in evaluating fault-mitigation designs. We are using the dynamical system modeling and simulation capabilities of Matlab®, Simulink®, Stateflow®, a tool-suite available from Mathworks Inc. to perform the simulation.

The simulation modeling begins with low-level hardware. We are developing low-level Stateflow simulation models of communication hardware protocols, along with the DMA processor that transfers data between memory and the communication link. In the model the network topology is captured by connecting communication ports. The kernel interface to the communication hardware is modeled as a communication channel. Stream structures, which implement communication conduits between software processes, are modeled on top of the channel, with multiple streams sharing a channel. The cyclic task scheduler is also modeled. Each of these models attempts to replicate the behavior of the kernel.

Finally, the application processes and interconnections are modeled with simplified behavior of the communications of each task. The next step in the simulation engine will be to add the fault mitigation simulation. We are also working to fully configure the simulation from the modeling environment, in terms of the hardware network topology, and the flow of data in the system.

11.8.3.2 Demonstration Project 2003

The RTES Collaboration created a demonstration for the 2003 Supercomputing Conference [29], which was held in Phoenix, AZ, in November 2003. The demonstration consisted of a prototype of the BTeV L1 trigger hardware (using 16 DSPs) running a prototype of the L1 trigger software. The software included fault handling tools (VLAs and ARMORs), and faults were injected into the system to demonstrate the response of the system and operation of the fault handling technology.

This demonstration project required that the research groups confront the issues needed to integrate their tools into a working system, and has allowed the groups to investigate the strategies and tools that have been developed so far. The next demonstration project (described below) will be developed using the prepilot L2/3 Farm.

11.8.3.3 Demonstration Project 2004

Having explored the issues associated with embedded processors in the L1 Trigger (Demonstration Project 2003), the RTES Collaboration is currently engaged in developing a commodity processor based model of the L2/3 Trigger. This new project will use the BTeV L2/3 prepilot system, a small (approximately 30 dual-CPU) farm of modest performance (333-500 MHz) Linux processors as a platform for understanding messaging, monitoring, and control in L2/3.

This demonstration project will be presented at an RTES workshop to be held in conjunction with the IEEE Real-Time and Embedded Technology and Applications Symposium in March, 2005. In addition to developing a sophisticated fault-adaptive model of the L2/3 trigger, this demonstration system will highlight the use of GME for defining and tailoring the underlying RTES runtime system software.

11.9 Production Plan

The main components that make up the BTeV trigger system are the following: Level 1 pixel trigger, Level 1 muon trigger, Global Level 1, L2/3 hardware and the L2/3 software. The production, production testing, and quality assurance of these components will be discussed in this document.

Item	Base Quantity	Additional	Total
L1 pixel pre-processor & segment tracker			
pixel pre-processors	480	48	528
segment trackers	480	48	528
L1 buffer readout links	480	48	528
optical link receivers	480	48	528
segment pre-processor interconnection	480	48	528
supervisor and link cards	480	48	528
L1 pixel trigger switch			
72-port commodity switches	8	0	8
HCA adpaters	128	0	128
HCA controllers	128	0	128
L1 pixel trigger farm			
commodity processors	264	0	264

Table 11.14: Pixel Trigger Components. The numbers of components listed under "Base Quantity" are those needed in the design. The "Additional" quantities are the extra components that are estimated to be needed to deliver a complete and operational system. These extra components include units needed for test stands by collaborating institutions for various test or development and to replace faulty or lost parts

11.9.1 L1 pixel trigger

Responsible institution: Fermilab

The Level 1 pixel trigger is the primary trigger for the BTeV experiment. The design, fabrication, and testing of the pixel trigger is divided into three phases. The first phase is the Pre-pilot phase that includes rudimentary hardware designs for every component in the pixel trigger architecture. The second phase is the Pilot system, which will operate as a complete trigger system since it consists of a complete trigger highway. The third, and final, phase is the Production pixel trigger system, which has a total of eight trigger highways. In this section we list all of the components that are part of the Production trigger system, and describe our plans for production, testing and quality assurance.

11.9.1.1 L1 pixel trigger electronic modules and interconnects

Electronic modules and interconnects used in the L1 pixel trigger system will be either commercial off-the-shelf (COTS) modules or in-house designs. The in-house designs will be assembled at Fermilab if the quantities are sufficiently small (typically, less than ten). Otherwise, they will be assembled by outside assembly companies. Fermilab is responsible for the L1 pixel trigger hardware and a large quantity of the muon trigger hardware. Consequently, Fermilab will make the majority of COTS purchasing decisions, and will pre-qualify board fabrication and assembly companies based on vendor interviews, site visits and/or previous experience with board and module orders for in-house designs.

11.9.1.2 Quality control procedures

All modules will have a specifications document that includes, at a minimum, the operational requirements, circuitry descriptions, schematics, board layout, assembly instructions, and bill-of-materials. These documents will be completed and reviewed before the module design is complete. The specifications documents can be modified at each stage throughout the Pre-pilot and Pilot testing phases as operational information is gathered. The Production specifications will not change after the production design has been reviewed and approved.

Each module design will be reviewed by BTeV engineers and physicists for safety, manufacturability, and compliance with requirements before purchase or production. All module designs will be required to meet BTeV, Fermilab, and OSHA safety standards for electrical systems and equipment. Design reviews for Pilot and Production stages will include a review of experience with previous Pre-pilot and Pilot modules.

All module fabrication contracts will include a clause specifying that the printed circuit board manufacturer will meet IPC standards for printed circuit board fabrication. All module assembly contracts will include a clause specifying that the assembly company will meet IPC standards for electronic assembly.

Incoming modules will be inspected for compliance with fabrication and assembly standards. 100% of pre-pilot and pre-production modules will be inspected. Based on past experience with manufacturing companies, 100% of pilot and production modules will be inspected or sampled at the 20% level to meet production schedules.

11.9.1.3 Testing procedures

Module testing will include a series of increasingly complicated tests. In-house designed modules that pass visual quality control inspection will be individually power tested by having nominal operating voltages applied and monitored for power consumption. This will confirm module power distribution and component placement. COTS modules will be similarly (power-only) tested. Power testing will be followed by functional tests to confirm stand-alone operation of each module. There may be limitations on which functions can be tested if a module's operation relies on a partner module. Power testing will be combined with functional testing as soon as the module design and assembly process (for in-house designs) are proven consistent. Interconnections with neighboring modules will be included, and then functions that depend on these interconnections will be tested. This process will gradually involve the support infrastructure, which is commonly referred to as a "test stand". At this point, the interconnecting cables will be included and tested. An increasing scope of combinations of modules will be tested until a partial system containing at least one complete data path is assembled. This partial system will be used to test system functionality and the inter-dependence of at least one instance of all the modules in the system. This is commonly called a "vertical slice" of the system. All modules for the Pilot system will proceed through individual module functional tests and vertical-slice system testing. Additional testing steps beyond the functionality needed for BTeV will be used to confirm the testing process as well as to fully test the design and operation of the modules and subsystems. Production testing will be done in the vertical slice environment that will be established at the Feynman Computing Center Integration Test Facility (ITF).

During production module testing, a database will be established and maintained in order to collect data that can be used to establish mean time between failure (MTBF) and mean time to repair (MTTR) statistics for the modules. This database can be an early indicator of reliability problems, component manufacturing problems and/or design weaknesses. It will also help to predict the number of spare components needed for the life of the experiment.

Each module or subsystem will have a user manual to provide a reference for experiment operators and technicians during the life of the experiment. The user manual will be a separate document or included in the specifications document. It will be reviewed for completeness and usefulness before system commissioning starts.

The commercial (COTS) processors and switches of the Level 1 pixel trigger are comparable to the Level 2/3 Farm components described in Section 11.9.4. As such, the Level 2/3 Quality Control procedures described in Section 11.9.4 will be followed for the COTS elements of the Level 1 pixel trigger.

11.9.2 Level 1 muon trigger

Responsible institution: University of Illinois

The Level 1 muon trigger will operate independently of the pixel trigger in that it generates trigger results based on the muon detector. The design of the muon trigger hardware is based on the design of the pixel trigger, and in many cases will employ identical hardware, but in smaller quantities. One specific example is the muon trigger farm. It will use the same processors, same power supplies, and the same packaging, but only one tenth as many components as the pixel trigger farm. In this section we list all of the components that are part of the production muon trigger system.

The plans for production, testing, and quality assurance for the Level 1 muon trigger are the same as those described for the Level l pixel trigger in Section 11.9.1 of this document, except that there are only pre-production and production phases; there is no pilot phase for the muon trigger. For the muon-trigger hardware that is identical to hardware in the pixel trigger, the pixel trigger group at Fermilab will be considered as the “vendor” for this hardware. As such, all production, testing and quality assurance will be conducted exactly as specified for the Level 1 pixel trigger.

For the remaining hardware that is specific to the muon trigger, the production, testing, and quality assurance will follow the Level 1 pixel trigger plans, employing the same manufacturers and Fermi-qualified contractors and executing the same procedures and protocols. Due to the proximity of Fermilab to the University of Illinois, it is easy to conduct much of

Item	Base Quantity	Additional	Total
Muon pre-processor subsystem			
muon pre-processors	48	5	53
optical link receivers	48	5	53
L1 buffer readout links	48	5	53
muon pre-processor interconnection	3	1	4
supervisor and link cards	48	5	53
L1 muon trigger switch	1†		1†
L1 muon trigger farm	1‡		1‡
Muon trigger supervisor and monitor	1‡‡		1‡‡

† The L1 muon trigger switch will be a scaled down version of the L1 pixel trigger switch.

‡ The L1 muon trigger farm will be a scaled down version of the L1 pixel trigger farm, using 24/264 (∼1/10) as many processors.

‡‡ The muon trigger supervisor and monitor will be a scaled down version of the pixel trigger supervisor and monitor, servicing 1/10 as many processors.

Table 11.15: Muon Trigger Components

this work at Fermilab so as to further leverage the similarities between the pixel and muon triggers.

The commercial (COTS) processors and switches of the Level 1 muon trigger are comparable to the Level 2/3 Farm components described in Section 11.9.4. As such, the Level 2/3 Quality Control procedures described in Section 11.9.4 will be followed for the COTS elements of the Level 1 muon trigger.

11.9.3 Global Level 1

Responsible institution: Fermilab

Global Level 1 (GL1) receives results from trigger calculations that are performed by the Level 1 pixel and Level 1 muon triggers. The results are processed by GL1 hardware in order to produce a Level 1 trigger decision for each beam crossing. GL1 hardware consists of processing nodes that are identical to the processing nodes used in the Level 1 pixel trigger. Here we list components that are needed for GL1.

The plans for production, testing, and quality assurance for GL1 are the same as those described for the Level 1 pixel trigger in Section 11.9.1 of this document.

Item	Base Quantity	Additional	Total
Global Level 1			
commodity processors	8	0	8
timing modules	8	2	10

Table 11.16: Global Level 1 Trigger Components

11.9.4 Level 2/3 hardware

Responsible institution: Fermilab

The Level 2/3 hardware consists of commodity PC's. These are operated as a collection of PC farms in the trigger system. The design, procurement, and testing of the L2/3 hardware will proceed in three stages. In the first stage, a Pre-pilot PC farm will be constructed from already existing hardware from old Computing Division PC farm nodes. The second stage consists of a Pilot farm that has all the components configured to act as a trigger system and will represent about 5% of the full Production system. In the third stage, the Production farm will be constructed and then integrated into the complete BTeV trigger system. The procurement and quality assurance procedures for the Level 2/3 hardware are described in this section.

11.9.4.1 Components of the Level 2/3 trigger hardware

The major cost components of the Level 2/3 hardware are listed in Table 11.17. All the items are expected to be commodity (COTS) items. The Farm worker PC's, Manager-I/O Host PC's, and storage units will all be subjected to the same procurement and quality assurance testing procedures. This consists of a selection and an evaluation stage to evaluate and qualify particular computer node products and vendors, followed by quality assurance testing after the nodes have been received.

11.9.4.2 Pre-pilot, Pilot and Production Level 2/3 Systems

A Pre-pilot L2/3 trigger PC farm has been built using 40 dual processor worker PC's and a dual processor Manager PC in FY04. The worker PC's are a mixture of old Fermilab Computing Division Farm PC's. These consist of a regular ATX desktop case each containing two 333 MHz Pentium II CPU's, or dual 500 MHz Pentium III CPU's. These are sufficient for development purposes. The worker nodes are networked via a 10/100 switch to a Manager PC. The Manager PC is connected to the switch via two 1000 Base-T gigabit Ethernet ports. This Pre-pilot PC farm is being used for a number of development projects for the L2/3 Trigger. It will be used to try out different infrastructure software to handle and manage farm worker processing, for developing infrastructure L2/3 Trigger and DAQ specific code, and testing and developing monitoring and control code from the RTES project. The

Item	Quantity
Farm worker CPU's	1536
L2/3 host PC's	72
L2/3 storage	1536

Table 11.17: Major L2/3 Hardware Components

Pre-pilot L2/3 Farm will also be used as one of the test beds for developing and testing software that will be needed to use the L2/3 PC farm as an offline processing resource, for example, Grid-related software to enable part of the farm for Monte Carlo simulations or data processing when processor cycles are free.

The Pre-pilot L2/3 farm will also be used to develop more reliable and manageable worker nodes, for example, by investigating the effect of diskless operation or using solid state disks for the OS and system related software, or other minor hardware modifications for monitoring purposes. The software development work will be carried out by the BTeV Trigger team and members of the RTES group, together with some consultation with the Fermilab Computing Division, including the Farms group, the CDF/D0 Reconstruction groups and the QCD Lattice engineering group.

A more powerful Pilot L2/3 PC farm will be built consisting of about 5% of the baseline farm (one full rack out of 24). These farms PC's will be used for continuing software development and coding. If blade servers are a viable alternative, the Pilot PC farm is likely to consist of this technology for evaluation and development.

Since the L2/3 Trigger farm and DAQ infrastructure will not be purchased until relatively late in the project, simulations will be used for queuing studies. These studies will be carried out in the first two years as part of the development work needed to decide on the L2/3 Trigger farm design and technologies. Even without a large PC farm, individual hardware components can be tested to ensure that specifications can be met. In addition the prototype farm can be used for system queuing studies. When the Pilot farm is available, these queuing and network throughput studies will continue on more up-to-date hardware, leading up to several data challenges closer to the end of the project's construction period.

11.9.4.3 Quality control and testing procedures

The major purchases for the L2/3 Trigger PC farm will go through a process similar to the Computing Division's purchase of the fixed-target, CDF, and D0 PC farms. The products and vendors will go through a thorough selection and evaluation stage after a detailed specifications and requirements document for the PC worker nodes has been produced and sent to vendors. This will enable vendors to perform the evaluation tests themselves. Sample nodes from prospective vendors will then be subjected to evaluation tests. The actual set of tests will be developed as we approach the time of purchase, when more is finalized about the L2/3 trigger. The current set of tests used by the FCC Farms Group consists briefly of running benchmarks and applications to reach a high level of CPU usage and disk access every 10 minutes, the disk is filled once a day and network transfers are performed periodically. These tests are performed continuously for 14 days. The Computing Division's OSS PC farm group test procedures are an evolving process and we intend to incorporate what is learned from their quality assurance testing. The current set of documents describing the evaluation process and giving the specifications for purchases and describing the evaluation tests for the Computing Division Run 2 farms can be found at: `http://www-oss.fnal.gov/scs/farms/acquisitions.html`. This web page can only be accessed from a computer on the Fermilab website. The relevant documents are included in the cost book.

Once the L2/3 farm nodes are received, they will be put through a set of tests similar to the evaluation tests, except that the Q&A tests will be run for a longer period of time. The Computing Division's OSS group found that some problems appear after many months of continuous operation. The Computing Division's OSS quality assurance testing is similar to the evaluation testing described in the documents above except that the nodes are required to be up and tested for 30 continuous days instead of 2 weeks. The tests are still evolving as more is learned about what long-term problems can arise. The tests consist of benchmarking; stress and long-term tests of memory, CPU, disks and networking; tests of other devices like floppy disk and CDROM drives; electrical and thermal characteristics monitoring; and downtime monitoring. Moreover, vendors are evaluated on their competence in the Linux operating system and in downtime incidents and other services provided during both the evaluation and quality assurance testing stages. We will continue to monitor the progress of the Computing Division's evaluation and quality assurance testing and start to participate in our own testing for a small pilot system. We will also develop the L2/3 system to be more reliable. For example, it has been found that most of the problems occur with hard disks, and we are investigating the possibility of using solid-state disks for system critical data.

11.9.5 L2/3 software

Responsible institution: Fermilab

The L2/3 trigger software consists of reconstruction software that uses Level 1 trigger output and raw data. The Level 2 trigger will detect the presence of secondary vertices, while Level 3 will reconstruct decay topologies and thereby the heavy-quark content (such as charm or beauty as distinguished from light-quark background) and the type of heavy-quark decay. Based on a set of predefined trigger tables and the reconstruction results the trigger will accept or reject a given beam crossing. At Level 2, only the pixel data and possibly some of the data from forward-tracking stations will be used. At Level 3, data from all BTeV detector systems will be used to refine the trigger selection criteria. The reconstruction algorithms at Level 2 include pattern recognition and track fitting. At Level 3, we include RICH particle identification, muon identification and electromagnetic-calorimeter shower reconstruction. The triggers will be configurable in that some predefined list of parameters will be downloaded onto processing nodes at the beginning of each run. For L2/3 software we will follow standard procedures to create and manage production versions of the software. The reconstruction software is not unlike the software that is typically written for offline computation, and as such it will be managed like any other BTeV offline package. We will therefore follow rules and conventions established for offline software. At this time we do not envision any problems with this approach. For instance, both the BTeV offline and online code is developed using the same industry-standard tool to implement version control (CVS repositories). Similarly, the bug-tracking systems that we intend to use should be the same between offline and online code development. We also do not foresee any boundaries between developers from different institutions as the software is designed, implemented, and tested. Therefore, everybody working on L2/3 software is likely to be involved in developing code, testing and quality assurance. Code development for Level 2 and Level 3 should be very similar, since the programming language is expected to be the same and the same software infrastructure will be used for both Level 2 and Level 3. Therefore our plans for production, testing, and quality assurance are the same for both.

11.9.5.1 Prototyping and preliminary production versions

The methodology that will be used for prototyping and preliminary production versions of software is the same as that used for any other software development for BTeV. The code must be maintainable by including documentation and by using version control during the development of the software. The code must be able to process Monte Carlo generated events, as well as real data coming from the experiment. The reconstruction programs will be able to produce their own diagnostic output in the form of histograms or n-tuples.

11.9.5.2 Production versions of code for the Level 2/3 farm

Production versions of code must be tested and verified offline, both on real data and Monte Carlo generated data. Executables of the code will be built using a standard BTeV scripting procedure, ensuring that we can rebuild executables at a later stage in the experiment. This implies that executables cannot be built using code that comes from someone's personal computer or private directory. Each Level 2/3 algorithm will be documented in a manner that describes the algorithm and its predicted performance on a given set of Level 1 data. This implies that prior to execution on the Level 2/3 farm one must have a predefined figure of merit against which one can benchmark the performance of new or different versions of algorithms.

When a new version of an algorithm needs to be tested (prior to implementation on the Level 2/3 farm), the new algorithm will be executed on a small portion of the data acquisition system (DAQ). This is also referred to as a DAQ partition If the algorithm is successful on a small scale, then the algorithm can be tested on a larger number of processing nodes (a larger partition). When this part of the test has been successfully completed, the algorithm can be deployed on all L2/3 processing nodes.

11.9.5.3 Maintenance, verification, and control of Level 2/3 configuration

Configuration files and trigger tables are critical items in the trigger system, since they determine the sharing of processing and network resources between different needs for the system. As such, they will be critically reviewed, circulated, discussed within the collaboration and finally approved by BTeV Management.

Bibliography

[1] For example, HERA-*B* and LHC-*b*; see talks by E. Gerndt and O. Schneider, Beauty '99 Conference, Bled, Slovenia.

[2] E.E. Gottschalk, "BTeV Detached Vertex Trigger", *Nucl. Instrum. Meth.* **A473** (2001) 167.

[3] BTeV Proposal, A. Kulyavstev *et al*, `http://www-btev.fnal.gov/DocDB/0000/000066/002/index.html`.

[4] Update to BTeV Proposal, G.Y. Drobychev *et al*, BTeV-doc-316.

[5] M. Wang, "BTeV Level 1 Vertex Trigger Algorithm", BTeV-doc-1179.

[6] M. Wang & J.Y. Wu, "Level 1 Trigger DSP Timing Studies and the Hardware Hash Sorter", BTeV-doc-3361.

[7] G. Cancelo *et al*, "The DSP-based Level 1 Trigger Track and Vertex Pre-prototype Hardware", BTeV-doc-3360.

[8] `http://www.altera.com/`

[9] `http://www.mathworks.com/`

[10] `http://www.mellanox.com/products/hpcperformance.html`

[11] `http://vmi.ncsa.uiuc.edu/index.php,`

[12] `http://lqcd.fnal.gov/benchmarks/newib/index.html`

[13] M. Bowden, H. Gonzalez, S. Hansen, A. Baumbaugh, "A high-throughput data acquisition architecture based on serial interconnects", IEEE Transactions on Nuclear Science, vol. 36, Issue 1, Feb. 1989; A. Booth, D. Black, D. Walsh, M. Bowden, E. Barsotti, "Simulation and modeling of data acquisition systems for future high energy physics experiments", IEEE Transactions on Nuclear Science, vol. 38, Issue: 2, Apr. 1991, Pages:316-321; T.M. Shaw, A.W. Booth, M. Bowden, H. Gonzalez, P.K. Sinervo, K.J. Ragan, M.S. Baker, C.C. Dong, R.K. Kwarciany, R. Van Conant, M.J. Whitman, "The one that shows

that something was actually built is: Architecture and development of the CDF hardware event builder", IEEE Transactions on Nuclear Science, vol. 36, Issue: 1, Feb. 1989 Pages:765-769.

[14] G. Cancelo, "Dataflow analysis in the L1 Pixel Trigger Processor", BTeV-doc-1178, Sept. 2002.

[15] M.H.L.S. Wang for the BTeV Collaboration, "The BTeV Trigger Architecture", *Nucl. Instrum. Meth.* **A511** (2003) 161.

[16] S. Cittolin *et al.*, in *Workshop on Recent Developments in High Energy Physics,* NCSR Demokritos, 9–11 April, 1998, p. 87.

[17] Associative memories (data-storage structures designed for rapid parallel search operations) are finding increasing use in industry; see for example the "content-addressable" memories made by MOSAID Technologies, `http://www.mosaid.com/networking/index.html`.

[18] See also D. Husby, "Systolic Associative Arrays for Track Finding," `http://www-ese.fnal.gov/eseproj/trigger/asmem/asmem.pdf`, and "Associative Memory for Track Finding," `http://www-ese.fnal.gov/eseproj/trigger/asmem/`.

[19] See for example M. H. Schub *et al.*, Nucl. Instr. Meth. **A376**, 49 (1996); W. Sippach, G. Benenson, and B. Knapp, IEEE Trans. Nucl. Sci. **NS-27**, 578 (1980); R. G. Cooper, IEEE Trans. Comp. **C-26**, 1123 (1977).

[20] D. Husby *et al.*, *Nucl. Instrum. Meth.* **A383** (1996) 193.

[21] W. Selove, in *Proceedings of the Workshop on B Physics at Hadron Accelerators*, P. McBride and C. S. Mishra, *eds.*, Fermilab-CONF-93/267 (1993), p. 617.

[22] D. M. Kaplan and V. Papavassiliou, in **CP Violation**, X.-H. Guo, M. Sevior, and A. W.Thomas, eds. (World Scientific, Singapore, 2000), p. 116; D. M. Kaplan, in *Proc. Symposium on Flavor Changing Neutral Currents: Present and Future Studies*, Santa Monica, CA, 19–21 Feb 1997, D. B. Cline, ed. (World Scientific, Singapore, 1997), p. 81.

[23] `http://www-btev.fnal.gov/`

[24] T. Elrad, R. Filman, and A. Bader, "Aspect-Oriented Programming," *Communications of the ACM*, vol. 44, no. 10, pp. 29-32, Oct 2001.

[25] "GME 2000, The Generic Modeling Environment," `http://www.isis.vanderbilt.edu/Projects/gme/`

[26] `http://www2.lm-sensors.nu/~lm78/docs.html`

[27] L. Picolli, "Evaluation of RTES components in a large scale DAQ," 13th IEEE-NPSS Real Time Conference, May 2003, Montral, Canada. http://www-btev.fnal.gov/cgi-bin/DocDB/ShowDocument?docid=1792

[28] http://www-btev.fnal.gov/public/hep/detector/rtes/

[29] http://www.sc-conference.org/sc2003/

[30] http://dspvillage.ti.com/docs/dspvillagehome.jhtml

[31] http://www.microsoft.com/homepage/default.htm

[32] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, Massachusetts, Addison Wesley, (1998).

[33] S. Tamhankar, J. Oh, and D. Mosse, "Design of Very Lightweight Agents for Reactive Embedded Systems," S. Tamhankar, J. Oh, Syracuse University, and D. Mosse, University of Pittsburgh, 10th IEEE *International Conference on the Engineering of Computer Based Systems*, April 2003, Huntsville, AL. http://www-btev.fnal.gov/cgi-bin/DocDB/ShowDocument?docid=1864

[34] K. Whisnant, Z. Kalbarczyk, and R. Iyer, "A Foundation for Adaptive Fault Tolerance in Software," in Proc. of Conference on Engineering of Computer Based Systems (ECBS) 2003.

[35] http://www.c2.com/cgi/wiki